Universidad Autónoma de Madrid
Escuela politécnica superior

Doble Grado en Ingeniería Informática y Matemáticas

Trabajo Fin de Grado

ESTUDIO Y ANÁLISIS DE ESQUEMAS DE VOTACIÓN ELECTRÓNICA MEDIANTE PROTOCOLOS DE FIRMAS DIGITALES ANÓNIMAS

Autor: Gonzalo José Abad-Pérez Espuelas
Tutor: Francisco de Borja Rodríguez Ortiz
Julio 2017

ESTUDIO Y ANÁLISIS DE ESQUEMAS DE VOTACIÓN ELECTRÓNICA MEDIANTE PROTOCOLOS DE FIRMAS DIGITALES ANÓNIMAS

Autor: Gonzalo José Abad-Pérez Espuelas
Tutor: Francisco de Borja Rodríguez Ortiz

Grupo de Neurocomputación Biológica
Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio 2017

Resumen

Resumen

En este trabajo se ha llevado a cabo un estudio del problema del voto electrónico analizando el estado del voto electrónico actualmente y haciendo un estudio de las posibles técnicas criptográficas que se podrían usar para resolverlo. Este trabajo se ha centrado en el estudio de las firmas ciegas y firmas grupales, como posible base de un protocolo de votación electrónico. Se procedió primero a estudiar las firmas ciegas y más adelante gracias a la capacidad de auditoría sobre el voto que aportan las firmas grupales el trabajo se centró sobre estas. Se diseñó de modo que haya auditoría individual realizada por el votante y global realizada por la administración.

Después de estudiar estas tecnologías se procedió a diseñar un protocolo de votado electrónico basado en 3 servidores independientes, a los que llamamos G1, TokenV Dispatcher y Voting Server. Los conceptos más relevantes de este protocolo son los siguientes:

- Estos servidores hacen uso del secreto dividido. Ningún servidor es capaz de averiguar por sí mismo la asociación votante y voto, sin embargo todos poseen información parcial que puesta en conjunto serviría para hacer una auditoría global.
- Se utiliza la técnica de la pseudonímia para enmascarar al votante y así conseguir anonimato. Esto solo para el protocolo desarrollado basado en firmas grupales.
- Las firmas grupales se utilizan para la identificación anónima de votante válido.
- Las firmas grupales facilitan la existencia de una auditoría individual del voto por parte mediante las pruebas de conocimiento cero de autoría sobre las firmas.
- Si las entidades de voto, es decir, las que forman parte activa del mismo, siendo estas G1 y TokenV Dispatcher, se ponen de acuerdo se puede realizar una auditoría global del proceso de votado.

Después de diseñar este protocolo se procedió a realizar un análisis tanto del anonimato como de la seguridad del esquema atendiendo a los mayores problemas conocidos que puede presentar un protocolo de voto electrónico. Recalcando posibles debilidades del sistema, y analizando las medidas tomadas para aumentar la seguridad frente a una serie de ataques.

Posteriormente se procede a implementar una prueba de concepto utilizando Python, MySQL y la librería libgroupsig ([1]). Se procede acto seguido a realizar una serie de pruebas y simulaciones de voto para probar el correcto funcionamiento de la implementación.

Palabras Clave

Privacidad, Anonimato, Pseudonímia, Voto Electrónico, Seguridad, Firmas Grupales, Protocolo, Trazabilidad, Revocación, Firma Ciega, Prueba de Conocimiento Cero, TOR, Secreto Dividido, Auditoría.

Abstract

This works involves a study of the problem of electronic voting, making an analysis of the state of electronic voting nowadays and making a study of the possible cryptographic technics that could be used to solve it. This work is focused on the study of blind signatures and group signatures, specially in the last one mentioned, as the possible base of a electronic voting protocol. This work first studied the blind signatures and ahead in the study, it focused more on the group signature thanks to the possibility to implement an individual vote accounting, this is the main reason why group signatures was chosen over blind signatures.

After the study of these technologies we proceeded to design an electronical vote protocol based in 3 independent servers, that we will call G1, TokenV Dispatcher and Voting Server. The most relevant concepts of this protocol are the following:

- This servers use the concept of shared secret. No server is capable to stablish an association between a voter and a vote by itself without colaboration of other servers.
- The technique of pseudonimity is used to mask the voter and gain anonimity.
- Group signatures are used for the anonymous identification of a valid voter.
- Group signatures grant the possibility of having an individual accounting that a voter can do to verify that the entire process of voting has been made correctly, this is granted thanks to the zero knowledge proofs.
- If the independent organisms collide and collaborate they can proceed to make a global accounting of the process of vote.

After designing this protocol we proceeded to make an analysis of the anonimity as well as an analisys of the security of the protocol, taking into account the biggest known problems of electronic voting and the most typical attacks to it. Pointing out possible weaknesses of the protocol and analysing the proactive measures that could be taken in order to reenforce it against this possible attacks.

After it an implementation of a proof of concept of the protocol was made using Python, MySQL and the library libgroupsig ([1]). Then the proof of concept was subject of several tests and simulations of voting in order to verify the correctness of the implementation.

Key words

Privacy, Anonimity, Pseudonymy, Electronical Voting, Security, Group Signatures, Protocol, Traceability, Revokation, Blind Signature, Zero Knowledge Proof(ZKP), TOR, Divided Secret, Accounting.

Agradecimientos

Quiero agradecer en este trabajo principalmente a Paco por haber sido tan atento y haberse mostrado tan interesado y activo en mi trabajo, también quiero agradecer a Mario y a Álvaro su colaboración y esas sesiones de brainstorming tan necesarias para encontrar vulnerabilidades e ir mejorando este protocolo poco a poco, y por supuesto agradecer a Jesús, el creador de la librería su trabajo ya que sin su librería libgroupsig no podría haber implementado este protocolo y por haberse mostrado siempre tan dispuesto a ayudar y tan interesado por el proyecto.

Índice general

Índice de Figuras **x**

Índice de Tablas **xii**

1. Introducción **1**

1.1. Motivación del proyecto 1

1.2. Objetivos y enfoque 2

1.3. Estructura de la Memoria 2

2. Estado del Arte **5**

2.1. Introducción 5

2.2. Privacidad y Anonimato 5

2.2.1. Privacidad 5

2.2.2. Anonimato 6

2.2.3. Integridad 6

2.2.4. Criptografía de Clave Pública 7

2.2.5. Firma Digital 7

2.3. Pruebas de Conocimiento Cero 9

2.4. Firmas Digitales Avanzadas 10

2.4.1. Firmas de Anillo 10

2.4.2. Firmas Grupales 11

2.4.3. Firmas Ciegas 12

2.5. Proceso de Votado 13

2.5.1. Voto Físico 14

2.5.2. Voto Electrónico 14

3. Sistema, diseño y desarrollo **21**

3.1. Protocolo de Voto Electrónico basado en firmas grupales 21

3.1.1. Análisis de Funcionamiento Correcto del Protocolo 25

3.1.2. Análisis de Seguridad del Esquema 27

3.1.3. Análisis de Anonimato del Voto 31

3.1.4. Auditoría del voto . . . 32
3.2. Requisitos y Casos de Uso . . . 34
3.2.1. Requisitos Funcionales . . . 34
3.2.2. Requisitos No Funcionales . . . 35
3.2.3. Casos de Uso . . . 36
3.3. Codificación . . . 36
3.3.1. Estructura del Sistema . . . 36
3.3.2. Bases de Datos . . . 37

4. Experimentos Realizados y Resultados **39**
4.1. Experimentos del sistema completo . . . 39
4.1.1. Pruebas de Funcionamiento Correcto . . . 39
4.1.2. Simulación de Voto . . . 40
4.1.3. Voto Múltiple . . . 41
4.1.4. Auditoría Individual . . . 41
4.1.5. Conclusiones . . . 42

5. Conclusiones y trabajo futuro **43**
5.1. Conclusiones . . . 43
5.2. Trabajo Futuro . . . 44

Glosario de acrónimos **45**

Bibliografía **47**

A. Conocimientos Adquiridos **51**

B. Manual de utilización **53**
B.1. Voting Client . . . 53
B.2. Server G1 . . . 53
B.3. Server A . . . 53
B.4. Server V . . . 54

C. Librería libgroupsig **55**

D. TLS/SSL **59**

E. Factorización de enteros en factores primos **61**

F. Teoría de Grupos **63**
F.1. La Teoría de Grupos en la Criptografía . . . 64
F.1.1. RSA . . . 64
F.1.2. Logaritmo Discreto . . . 64
F.1.3. Cifrado de ElGamal . . . 65

G. Suposiciones Criptográficas **67**
G.1. Suposiciones Criptográficas sobre Funciones Hash . . . 67
G.2. Decisional Diffie-Hellman . . . 67
G.3. Strong-RSA . . . 68
G.4. q-Strong Diffie Hellman . . . 68

H. Ejemplo de Prueba de Conocimiento Cero de la Cueva **69**

I. Prueba de Conocimiento Cero del Logaritmo Discreto **71**

J. Protocolo de Votado empleando Firmas Ciegas **73**
J.0.1. Firmas Ciegas Aplicadas al Voto Electrónico . . . 73

K. SQL Injection **79**
K.1. Medidas de seguridad . . . 80

L. DoS/DDoS **81**

M. Man in Middle **83**
M.1. Explotación . . . 84
M.2. Prevenciones . . . 84

N. TOR(The Onion Route) **85**
N.1. Funcionamiento . . . 85
N.2. Anonimato . . . 86
N.3. Servicios Ocultos . . . 86

Ñ. Medidas de tiempos de esquemas de firmas grupales de libgroupsig **87**

O. Código **93**
O.1. Código de Cliente . . . 93
O.2. Código Server G1 . . . 100
O.3. Código Server A . . . 106
O.4. Código Server V . . . 111
O.5. Código Protocol Codes . . . 116

Índice de Figuras

2.1. Proceso de Votado.(Fuente: *www.wikipedia.org*) 16

3.1. Protocolo de Voto Electrónico usando Firmas Grupales 23
3.2. Auditoría Individual del Voto 33
3.3. Diagrama de Casos de Uso 36

4.1. Simulación de Voto Satisfactorio 41
4.2. Tiempo de Voto Erróneo por Multiplicidad 41
4.3. Tiempo de Auditoría Individual 42

D.1. TLS Handshake (Fuente: *www.wikipedia.org*) 60

H.1. Probador entra por un camino hasta el fondo mientras Verificador espera fuera(Fuente: *www.wikipedia.org*) 70
H.2. Verificador entra hasta la bifurcación y elige un camino(Fuente: *www.wikipedia.org*) 70
H.3. Probador usa su conocimiento para satisfacer la pregunta(Fuente: *www.wikipedia.org*) 70

J.1. Voto electrónico usando Firmas Ciegas 74

L.1. Ataque DDoS (Fuente: *www.wikipedia.org*) 82

M.1. Ataque Man in the Middle(Fuente: *www.wikipedia.org*) 83

N.1. Ejemplo de comunicación de la red TOR (Fuente: *www.wikipedia.org*) 85

Ñ.1. Medidas de tiempo de Join 87
Ñ.2. Medidas de tiempo de Sign 88
Ñ.3. Medidas de tiempo de Claim 88
Ñ.4. Medidas de tiempo de Claim_Verify 89
Ñ.5. Medidas de tiempo de Open 89
Ñ.6. Medidas de tiempo de Reveal 90
Ñ.7. Medidas de tiempo de Trace_KTY04 90
Ñ.8. Medidas de tiempo de Trace_BBS04 91
Ñ.9. Medidas de tiempo de Trace_CPY06 91

Índice de Tablas

3.1. Tabla de la Base de Datos de G1 37
3.2. Tabla de la Base de Datos de TokenV Dispatcher 37
3.3. Tabla de la Base de Datos de Voting Server 37

1 Introducción

1.1. Motivación del proyecto

Desde los orígenes de la criptografía el ser humano ha mostrado interés en poder guardar secretos, con el avance matemático y tecnológico nuevas necesidades han surgido en el mundo criptográfico. Ya no solo interesa poder guardar algo oculto, sino saber a priori si ha sido alterado independientemente de poder leer el documento o no. Más tarde surge la necesidad de crear cifrados asimétricos dando lugar a la criptografía de clave pública, y muchos otros ejemplos que dan paso a muchas otras tecnologías revolucionaria. La criptografía es hoy de un uso fundamental en muchos ámbitos, véase en redes sociales, en la banca online, cloud computing, las comunicaciones seguras sobre red tipo Skype o VoIP y un largo etcétera. Sin duda la criptografía ha ido avanzando y cobrando mayor importancia en la medida que la tecnología se ha desarrollado, y uno de los ámbitos en los que la criptografía es protagonista es en el de voto electrónico.

En la sociedad moderna el voto forma parte básica de muchas organizaciones, como puede ser un gobierno democrático, una empresa, o una universidad. Sin embargo el voto electrónico no se ha convertido en un estándar, esto es debido a muchos factores, pero las ventajas de el voto electrónico son muchas, entre todas ellas destacan las siguientes:

- Posibilidad de votar sin desplazarse.
- Sistema más fiable y auditable que el voto físico.
- Contado de votos más fiable y rápido.
- Se reduce el margen de error por fallo humano.
- Se pueden permitir periodos de votación más largos.
- Se pueden realizar un mayor número de votaciones.

Por lo que el voto electrónico se considera algo muy deseable en la mayoría de los casos.

Muchos son los protocolos de voto electrónico que existen hoy día, desde esquemas enfocados a países enteros, como puede ser el que utiliza Brasil, a esquemas de votado que puede

utilizar cualquiera online para votaciones de carácter más reducido como Agoravoting. Las tecnologías utilizadas en estos diferentes esquemas son muy variadas, cada una con sus ventajas e inconvenientes, entre ellas destacar las firmas ciegas [2], las firmas grupales [3], los contadores criptográficos [4], o las mix networks [5]. El objetivo de este trabajo es, realizar un estudio de cómo utilizando algunas de estas se podría llegar a conseguir un protocolo de votado válido. Para acto seguido implementar una prueba de concepto y comprobar el correcto funcionamiento del mismo.

Una de las mayores motivaciones de este proyecto es el poder realizar algo útil, algo que pueda servir, en este caso sería realizar un protocolo de votado que pueda utilizarse, por ejemplo, en esta universidad.

1.2. Objetivos y enfoque

Los objetivos de este trabajo es de investigar las posibles implementaciones del voto electrónico en un entorno cerrado, como puede ser una pequeña empresa o una universidad. Se asume este escenario dado que el proceso de registro de los usuarios autorizados para votar, en este trabajo se asume este escenario por la potencial existencia de correos electrónicos institucionales u otro método identificativo similar. Los cuales se utilizan como identidad digital del votante y se le puede transmitir de forma segura el protocolo de votado. De no ser así se debería incluir en todo esquema de votación la fase de registro de votantes, que es un problema de la identidad digital, para este trabajo supondremos esa parte hecha ya por lo explicado anteriormente. Los objetivos de este trabajo son los siguientes:

1. Realizar un estudio de los sistemas de voto físico y electrónico actuales comparando estos esquemas y comprobando que se cumplen todas las bases deseables de un voto justo, anónimo y auditable.

2. Realizar un estudio sobre las firmas ciegas y los esquemas de cifrado de grupo y de qué protocolos pueden ser deseables para el voto electrónico, estudiar que ventajas aportan las firmas grupales y firmas ciegas al voto electrónico.

3. Realizar un análisis tanto de ciberseguridad de los esquemas como de anonimato imponiendo medidas proactivas para fomentar estas cualidades.

4. Realizar una prueba de concepto de uno de estos esquemas implementándolo.

5. Verificar la correctitud de la implementación mediante la simulación de votado bajo diferentes condiciones.

1.3. Estructura de la Memoria

En esta memoria se recoge el estudio y los objetivos del trabajo realizado en las siguientes secciones:

1. **Estado del Arte:** En esta sección se realiza un estudio de las firmas digitales avanzadas como las firmas ciegas y firmas grupales así como el problema del voto tanto físico como electrónico y las aportaciones que estas firmas pueden hacer al problema del voto.

2. **Sistema Desarrollado:** En esta sección se diseñan esquemas de voto electrónico basándose tanto en firmas ciegas como firmas grupales y se procede a un análisis de la seguridad y privacidad de estos protocolos. Acto seguido se realiza una prueba de concepto de el protocolo de firmas grupales mediante una implementación a pequeña escala del protocolo.

3. **Experimentos:** En esta sección se recogen pruebas realizadas sobre la implementación del voto electrónico mediante una serie de simulaciones en entorno cerrado con sus correspondientes resultados.

4. **Conclusiones:** En esta sección se recogen las conclusiones a las que se llegan después de realizar este trabajo, así como una reflexión de posible trabajo futuro sobre este proyecto para hacer un trabajo más completo y, posiblemente, llegar a implementarlo en una institución o universidad.

5. **Anexos:** Se recoge aquí información adicional proporcionada para la correcta comprensión del protocolo así como de las partes más internas que lo conforman.

2

Estado del Arte

2.1. Introducción

Con el paso del tiempo y la evolución de la tecnología cada vez más tareas de ámbito físico pasan al ámbito electrónico, hoy día las comunicaciones se realizan en casi su totalidad mediante internet por medio de las redes sociales, aplicaciones de mensajería instantánea, o aplicaciones de voz sobre IP. Sin embargo el voto no se ha adaptado del todo a el ámbito electrónico, y esto es dada la complicación de la implementación de un voto que ofrezca las garantías del voto físico implementado electrónicamente. Como veremos ya existen métodos de voto electrónico en el mundo como en Suiza, Austria o Brasil [6], se estudiaran en este trabajo técnicas de voto electrónico, para observar cuales son las posibles implementaciones del voto y estudiar que garantías ofrecen [7]. En esta sección se estudiará la evolución de la criptografía y los esquemas de voto actuales tanto físico como electrónicos para buscar una solución a este problema.

2.2. Privacidad y Anonimato

La privacidad y el anonimato han sido puntos clave desde los inicios de la historia de Internet, y estos conceptos son fundamentales especialmente importantes a la hora de desarrollar un sistema de voto electrónico.

2.2.1. Privacidad

Privacidad es la capacidad que un miembro A de una comunicación tiene de enviar un mensaje a B de modo que sabe que B podrá leer este mensaje pero si un atacante Z lo intercepta no será capaz de inferir ningún tipo de información del mensaje ni comprenderlo, solo si la clave de este mensaje se ha guardado efectivamente de un modo secreto. Esto se consigue generalmente mediante la criptografía simétrica, algunos de los algoritmos de cifrado simétrico más importantes son por ejemplo el DES, su sucesor más seguro el Triple DES (ver [8]), y el AES (ver [9]).

Los algoritmos de clave simétrica tienen su fuerza en el secreto de la clave, en el mundo moderno si dos miembros de la comunicación ya poseen un secreto común entonces usando el

algoritmo apropiado la transmisión del mensaje es segura, sin embargo internet es un mundo de comunicaciones muy globalizado, y es poco probable que los miembros de la comunicación posean ya un secreto compartido, es por ello que surge la criptografía Asimétrica, que será explicada más adelante.

2.2.2. Anonimato

El anonimato es la capacidad de ser participe de un sistema o proceso sin revelar la identidad del participante. Últimamente el anonimato ha cobrado mucha más importancia gracias a internet. La constante amenaza de ciberataques, la monitorización del tráfico de internet realizada por los ISP o por los gobiernos como bien reveló Edward Snowden sobre el programa de la CIA Prism [10].

Una de las tácticas más comunes hoy día realizadas por los usuarios se consigue mediante el uso de VPNs, las cuales utilizan la criptografía simétrica para cifrar el tráfico entrante y saliente de un ordenador que se envía todo a un servidor generalmente de otro país, y es este servidor el que realiza la petición como tal, de este modo, los ISP no pueden deducir que tipo de tráfico está usando su cliente. Otra técnica popular es la del uso de la red TOR (Ver Anexo N), cuya infraestructura anonimiza en gran medida el tráfico de los usuarios de la red haciendo rebotar su señal por múltiples nodos haciendo su señal no rastreable.

El anonimato es una parte fundamental del voto, ya que sin él, un votante podría temer las consecuencias de que el resto de votantes conozcan su opinión y puedan tomar represalias. Es por ello que el voto debe ser 100 % anónimo, para que los votantes puedan expresar su opinión de forma libre. Como dijo Oscar Wilde: "Dale una máscara a un hombre, y te dirá la verdad".

2.2.3. Integridad

La integridad es la capacidad de asegurarse que un documento permanezca invariante, y que en caso de cambiar se pueda detectar de forma veraz, y si pudiese arreglarse que sea arreglado. Esta noción es importante en criptografía ya que interesa saber si se puede haber intentado atacar una comunicación, esto también es importante para el voto electrónico, ya que no se puede permitir que los votos realizados puedan ser modificados por un atacante, esto en criptografía es posible verificarlo mediante las llamadas funciones hash (ver [11]).

Funciones Hash

Las funciones hash, también llamadas funciones resumen, generan un identificador unívoco del fichero, generalmente y hoy en día, no muy grande (En torno a 256 o 512 bytes) que identifican de forma unívoca(en probabilidad) el fichero dado.

La clave del éxito de las Funciones Hash reside en que son funciones de un solo sentido, se puede calcular de forma sencilla el hash de un fichero pero no se puede obtener ni el fichero a partir del hash ni información del propio fichero. Es trivial ver que esta función no es inyectiva, sea F una función hash (ver [11]) tal que la longitud en bits es de n bits, tenemos entonces 2^n posibles resultados, generamos $2^n + 1$ ficheros de bytes aleatorios, y les aplicamos la función hash, por el Principio de Palomar existen 2 ficheros con el mismo hash. Pero si no son inyectivas, ¿Por qué sirven para identificar de forma unívoca el fichero? Esto se consigue teniendo en cuenta que encontrar estos dos ficheros con el mismo hash es computacionalmente inviable.

Gracias a las funciones Hash si se quiere enviar un fichero a otro usuario y se adjunta el hash de forma segura, el receptor podrá comprobar que el fichero recibido no ha sufrido

modificaciones. Las funciones Hash más populares son SHA256 SHA512 diseñados por la NSA [11] o MD5 diseñado por Rivest [12].

2.2.4. Criptografía de Clave Pública

La criptografía de clave pública define un hito en la historia de la seguridad, poniendo solución a uno de los grandes problemas de la criptografía. Ahora, dos miembros que no compartan un secreto son capaces de comunicarse de forma segura.

En criptografía de clave pública se asume que un miembro de la comunicación posee dos claves, su clave pública, a la que llamaremos PU_A y su clave privada, a la que llamaremos PR_A. Estas dos claves conmutan en la composición de sus aplicaciones y la composición de ambas da la identidad, i.e. $\forall M PR_A(PU_A(M)) = PU_A(PR_A(M)) = M$.

La suposición de seguridad reside en que, la clave privada la conozca solo el individuo dueño de la misma, y que la clave pública pueda ser conocida por todos, como sus nombres indican.

Algoritmos de Clave Pública

A continuación se presentan 2 de los criptosistemas más conocidos de Clave Pública.

RSA

RSA es probablemente el criptosistema de clave pública más utilizado, fue desarrollado en 1977 por RSA Laboratories [13].

La fuerza del criptosistema RSA reside en la dificultad de factorizar números grandes en factores primos como viene explicado en el Anexo E, este es un famoso problema NP y hasta la fecha no se conoce un algoritmo eficiente para calcularlo. El RSA se basa en el teorema generalizado de Euler y la potenciación de números en grupos cíclicos para funcionar, una explicación más detallada viene dada en el Anexo F. Al estar basado en la potenciación de grandes números RSA es un algoritmo muy lento generalmente y suele utilizarse para el cifrado de pequeños contenidos, como una pequeña clave simétrica a usar, el hash de un fichero o documentos pequeños.

Curvas Elípticas La criptografía de Curvas Elípticas [14] es un esquema de criptografía de clave pública basado en el comportamiento de las curvas elípticas sobre campos finitos. Es un esquema muy innovador y expertos afirman que es el futuro de la criptografía de clave pública ya que ofrece la misma seguridad que otros esquemas como RSA usando claves de tamaño mucho menor.

2.2.5. Firma Digital

La motivación de la Firma Digital es la de obtener una equivalencia digital a la firma manuscrita, De aportar un acuerdo o conformidad de una persona con un documento, como claro ejemplo tenemos el de los contratos laborales o las peticiones de préstamos bancarios y un largo etcéterca. Este concepto es importante en esquemas de votación ya que puede ser utilizado para la fase de registro de votantes y la de distribución de claves. Además, el proceso de la firma digital es ampliamente utilizado por autoridades certificadoras, que forman parte fundamental del protocolo TLS/SSL. El cual se utiliza ampliamente en Internet, y en este caso, en los protocolos de votado.

Firma Digital Individual Veamos que es lo que ofrece la firma digital manuscrita:

- Se puede comprobar que la firma pertenece a una persona, en este caso por medio de grafólogos.
- Se puede comprobar que no ha sido falsificada.
- Se puede verificar que el documento no ha sido alterado, dado que está escrito con tinta no borrable sobre papel.
- El documento puede ser leído por cualquiera.

¿Cómo se lleva a cabo generalmente este esquema al mundo digital? Veámoslo con una implementación típica de firma digital utilizando RSA.

RSA Una firma digital basada en RSA consiste en los siguientes pasos:

- Generar un Hash de el documento a firmar digitalmente, ya que como hemos mencionado anteriormente el Hash es un identificador unívoco del documento y es de pequeño tamaño, cosa buena para la firma ya que la criptografía de clave pública es lenta.
- Cifrar el Hash generado con la clave privada del firmante.
- Enviar como documento, siendo M el documento $M||PR_A(Hash(M))$.
- El receptor calcula $\phi = Hash(M)$ y utiliza la clave pública de A y se la aplica a el adjunto del mensaje resultando $\psi = PU_A(PR_A(Hash(M))) = Hash(M)$.
- B compara si $\phi = \psi$ En caso contrario rechaza la firma digital.

¿Por qué es fuerte la firma digital?

- Si un atacante modifica el mensaje claramente $\phi \neq \psi$ Por lo que se rechaza la firma
- Si un atacante intenta generar la firma como tal no puede pues no posee PR_A.

Sin embargo uno de los problemas principales en la firma digital es la de asegurar que una persona es la dueña de una identidad digital, en este caso nuestra identidad digital es equivalente a nuestra clave pública, pero la generación de una clave pública la puede hacer cualquier persona, ¿Cómo sabemos que la persona que tiene la clave PR_A y por tanto le corresponde la clave pública PU_A es quien dice ser? Un enfoque que da respuesta a este problema es el de las autoridades certificadoras que es el más común.

Certificados Digitales y Autoridades Certificadoras

Una autoridad certificadora es una organización que atestigua que una persona es la que se posee una determinada clave pública. La robustez de los certificados digitales es tan fuerte como la confianza puesta en la autoridad certificadora, es por esto que una autoridad certificadora podría ser por ejemplo un gobierno, que asegura que la clave pública de una persona de su carnet de identificación corresponde efectivamente a esta persona en particular. Las autoridades certificadoras lo que otorgan es la comprobación de la biyección entre clave pública y persona/organización física. Pero ¿Cómo se consigue llevar a cabo? Mediante el certificado digital, una posible implementación es mediante el esquema de PKI's X.509 [15] [16], veámos ahora como se genera un certificado digital:

- La persona/organización rellena información de contacto relevante, nombre, NIF, Ubicación, correo electrónico, dominio de correos, y un largo etcétera, en resumen, su identidad digital. Y al final su clave pública.
- La autoridad certificadora comprueba la veracidad de esos datos y la correspondencia clave pública persona/organización.
- La autoridad certificadora firma digitalmente todo este contenido con su clave privada.

Por lo explicado anteriormente si todos confiamos en la autoridad certificadora podremos confiar en que una persona con un certificado digital es quien dice ser.

2.3. Pruebas de Conocimiento Cero

Las pruebas de conocimiento son una serie de protocolos por la cual un miembro de la comunicación demuestra que posee la solución a un problema sin revelar nada más que la veracidad del conocimiento de la solución, estas pruebas pueden utilizarse para probar pertenencia a un grupo de identidades [17] sin revelar la identidad del usuario. Este problema se puede utilizar, como veremos, en el contexto de la votación, para probar la pertenencia al grupo de votantes autorizados. Estos problemas son generalmente de índole matemática, y como veremos el concepto demostración de conocimiento de la solución de un problema no es el convencional, sino que se concibe como demostración el que la probabilidad de que no posea la solución al problema sea muy pequeña, es decir, la esté adivinando por azar.

Este tipo de pruebas son muy importantes ya que se pueden utilizar en criptografía de firmas grupales, ya que son utilizadas para reclamar la autoría de una firma grupal sin revelar la identidad del emisor [17].

En las pruebas de conocimiento cero hay 2 miembros que participan, el probador, que es quien quiere probar que posee un secreto, y el verificador, que es a quien el probador intenta convencer de que posee el secreto.

Según el nivel de intervención del verificador en el proceso existen 2 tipos de pruebas de conocimiento cero: Las interactivas y las no interactivas [18]. Las interactivas requieren una participación activa por parte del verificador, mientras que las no interactivas solo requieren una supervisión por parte del verificador.

Un probador honesto es aquel que realmente posee el secreto y quiere demostrar esa posesión, y un verificador honesto es aquel que quiere verificar de forma veraz la posesión de ese secreto.

Las pruebas de conocimiento cero deben satisfacer las siguientes condiciones:

- **Totalidad**: Si el probador y el verificador son honestos, y esto quiere decir que ejecutan cada uno su algoritmo de forma correcta y legítima, el probador convencerá al verificador de la veracidad de la posesión de la solución en probabilidad. ¿Esto que quiere decir? Que la probabilidad de que no la posea sea insignificante y generalmente matemáticamente improbable, esta cantidad puede variar según el algoritmo a usar.
- **Solvencia**: La probabilidad de que un probador deshonesto pueda convencer de que posee la solución al problema a un probador honesto es matemáticamente insignificante.
- **Conocimiento Cero**: Un verificador deshonesto no obtiene ningún tipo de información del problema además de que el probador demuestra que posee la solución al problema.

Si se cumplen las dos primeras condiciones se dice que el protocolo es una prueba de conocimiento, si además verifica la tercera se considera una prueba de conocimiento cero.

Es importante notar que las demostraciones de las que hablamos anteriormente no son en el estricto sentido matemático de verdad irrefutable, dado que esto es solo posible si el verificador obtuviese la solución como tal del problema. Estas demostraciones son demostraciones en probabilidad, cuando el probador da una respuesta al verificador siempre existe una probabilidad de que le esté engañando y se haya inventado la respuesta ya que el probador no la conoce. sea, en notación de probabilidad condicionada, A=P(El probador da una respuesta inventada consistente a la pregunta | Pregunta del verificador). Al ser una probabilidad vemos que si el probador hace sucesivas preguntas esta probabilidad va disminuyendo, i.e. $\lim_{n\to \text{inf}} A^n \to 0$ según el protocolo se establece una probabilidad de confianza, como ejemplo en matemáticas B se considera un suceso imposible $\Leftrightarrow P(B) < 10^{-50}$. Un buen ejemplo de estas pruebas viene dado en el Anexo H.

2.4. Firmas Digitales Avanzadas

Además de las firmas digitales convencionales existen una serie de firmas más avanzadas que nos van a permitir llevar a cabo tareas más complejas como puede ser firmar como un miembro de un grupo sin revelar la identidad del firmante, existen varios enfoques de este problema, a destacar las firmas de anillo y las firmas grupales que vienen explicadas a continuación.

2.4.1. Firmas de Anillo

Las firmas de anillo son un tipo de firmas en las que los usuarios son capaces de emitir firmas en nombre del grupo y un receptor es capaz de verificar que la firma grupal viene del grupo [19].

Lo que caracteriza a las firmas de anillo es que no hace falta una autoridad central para su utilización. Es decir, no hace falta una autoridad para la creación del grupo ni para añadir a alguien al grupo de cifrado. De hecho no hace falta si quiera el consentimiento de los miembros de un grupo de anillo para generar una firma de anillo con sus claves. Tan solo es necesario que la entidad que quiera emitir la firma de anillo conozca las claves públicas de las personas que desea que estén en el grupo.

Los requisitos para que un usuario genere una firma de anillo son, su clave secreta, y la clave pública de los miembros del grupo. Para que una entidad compruebe la validez de la firma de anillo de un mensaje, solo necesita la firma del mensaje, el mensaje y la lista de claves públicas de los miembros del grupo.

Requisitos

- Existe una lista de usuarios G de n usuarios, a este grupo G lo llamaremos el Grupo Anillo.
- Cada uno de estos usuarios posee una clave pública y una clave privada con su correspondiente firma digital con cada una de estas claves.
- Existe una función de cifrado simétrico $E(k, x)$ con su correspondiente función de descifrado $E^{-1}(k, x)$.
- Existe una función Hash h resumen de longitud l. Además se estandarizan las funciones f_s a un dominio en el que produzcan una salida de misma longitud.

2.4.2. Firmas Grupales

Las firmas grupales son un tipo de firmas digitales avanzadas en las que se puede comprobar que una firma viene de un conjunto de anonimato(el grupo) y no se puede extraer información del emisor a partir de la firma(salvo en circunstancias especiales en algunos esquemas por parte del administrador) [3].

Estas firmas son parecidas a las de anillo ya que al final ambas generan firmas que se puede comprobar que pertenecen a un determinado grupo, sin embargo poseen diferencias importantes, estas diferencias son las siguientes:

- En las firmas grupales se emiten identidades de miembro de grupo por parte de una autoridad mientras que en las firmas de anillo no se necesita de emisión, solo tener una firma pública/privada y que los miembros del grupo tengan firma también.
- Un grupo de firmas grupales es inmutable salvo la inserción de nuevos miembros por parte de la administración. Mientras que en las firmas de anillo no hace falta que los miembros del grupo autoricen al firmante a usar su firma ya que el firmante utiliza sus claves públicas.
- Algunos esquemas de firmas grupales se pueden abrir en el sentido de revelar la identidad de el emisor de una firma (Solo el Administrador) mientras que en las de anillo no se puede.
- Algunas de las firmas grupales cuentan con funciones más avanzadas como la revocación o la trazabilidad. Funciones que se explicarán más adelante.

Las firmas grupales deben cumplir los siguientes requisitos:

1. **Completitud:** Todas las firmas grupales emitidas por miembros válidos del grupo son verificables y su verificación es siempre correcta, todas las firmas grupales inválidas siempre dan verificación incorrecta.
2. **Exactitud:** Solo los miembros válidos del grupo pueden emitir firmas correctas
3. **Anonimato:** Dado un fichero y su firma grupal es imposible extraer la identidad del emisor de la firma sin poseer la clave de administrador.
4. **Revocabilidad:** Dada una firma grupal correcta, el administrador y solo el administrador es capaz de revelar la identidad del emisor de la firma.
5. **Trazabilidad(Opcional):** Dada una lista lista de usuarios revocados, un miembro del grupo puede verificar si la firma viene de el conjunto de anonimato de usuarios revocados sin revelar la identidad del emisor de la firma [20].
6. **No Linkabilidad:** Dadas 2 firmas grupales correctas, no se puede determinar si estas firmas han sido emitidas por el mismo miembro o no.
7. **No Reproducible:** No se puede emitir una firma válida de un miembro de grupo sin la participación voluntaria del mismo, ni aunque todos los otros miembros del grupo se pongan de acuerdo.
8. **Trazabilidad no Reproducible:** El administrador de grupo no puede acusar a un miembro del grupo de emitir una firma que el miembro no ha creado.
9. **Resistencia a Coalición:** Un subconjunto de usuarios válidos no pueden agruparse de modo que puedan generar firmas válidas y el administrador no pueda linkar esa firma con la identidad del emisor.

Veamos ahora algunos esquemas de firmas grupales.

Firmas Grupales Revocables Como hemos explicado anteriormente la revocabilidad [21] es la capacidad de un administrador de grupo de marcar a un miembro como revocado, de modo que las firmas que emita este miembro serán marcadas como revocadas. De modo opcional se puede revelar la identidad del autor. Las firmas que hacen uso de esta cualidad de la librería libgroupsig del artículo [1] son BBS04 KTY04 y CPY06, que vienen explicadas a continuación.

KTY04 KTY04 [20] es un esquema de firmas revocables que viene implementado en la librería libgroupsig. En este sistema la administración es la que realiza la revocación de los usuarios del grupo.

Este sistema se basa en una prueba de conocimiento cero que depende de varias suposiciones criptográficas, siendo estas la Diffie-Hellman Decisional (Ver Anexo G.2), el problema del Logaritmo discreto (Ver Anexo F.1.2) y la suposición Strong-RSA (Ver Anexo G.3).

Este sistema de los de dentro de libgroupsig es el sistema computacionalmente más costoso (Ver Anexo Ñ) ya que este utiliza de 17 a 19 exponenciales en funciones como firmar o verificar que una firma viene del grupo además de que genera firmas muy grandes ya que superan el 1KB. Sin embargo es el esquema que menos recursos requiere a la hora de generar el grupo.

CPY06 CPY06 [22] es un esquema que se basa en el problema q-Strong Diffie Hellman (Ver Anexo G.4) para la prueba de conocimiento cero en la que se basa. CPY06 es el sistema que mejor crece junto a BBS04 respecto al resto en cuanto a tiempo Ñ de ejecución respecto a los otros esquemas estudiados.

BBS04 BBS04 [23] es un esquema muy similar a CPY06 que, al igual que CPY06 se basa en la suposición q-Strong Diffie Hellman (Ver Anexo G.4) pero que sin embargo no propicia métodos de Claim y por consecuente de ClaimVerify, por contra es el esquema implementado que produce las firmas más cortas, alrededor de 200 Bytes por firma, lo cual hace a este esquema ideal para utilizar en protocolos de comunicaciones muy pesados.

2.4.3. Firmas Ciegas

Las Firmas Digitales Ciegas son firmas que permiten a una organización firmar un mensaje sin poder obtener ninguna información de lo que están firmando. Estas firmas fueron creadas por David Chaum [2].

Para hacer una pequeña introducción a cómo funcionan las firmas digitales ciegas vamos a explicarlo con una analogía al mundo real.

- El votante toma su papeleta de votación y la envuelve en papel calco opaco para que no se pueda ver el interior, lo sella para que no se pueda abrir, para luego enviarlo en un sobre con su remitente al comisario de la votación.
- El comisario comprueba el remitente y comprueba si esta persona está autorizada a votar y no ha votado antes, de ser así firma sobre el papel de calco, es decir, emite su firma sobre algo que no puede leer. Acto seguido envía el paquete al remitente.
- Para el día de las elecciones, cada votante envía en un sobre sin remitente su voto al comisario, el comisario podrá comprobar que el voto es válido pues su firma estará sobre el voto, y será anónimo pues el sobre no lleva remitente.

Con esas ideas y siguiendo la analogía explicada existe un mecanismo de papel de calco, un mecanismo que impide ver su interior pero que al escribir por encima la tinta se transmita y firme el contenido del interior, esto matemáticamente lo podemos conseguir si la función simétrica descrita con anterioridad conmutase con la función de firma del comisario. Siendo más formales, expliquemos los detalles de funcionamiento de una firma digital ciega [24]:

- Sea $E(k, x)$ una función de cifrado simétrico y $E^{-1}(k, x)$ su inversa donde k es la clave y x el mensaje.
- Sea $PR_A(x)$ La función de cifrado usando la clave privada, y $PU_A(x)$ la función de cifrado mediante su clave pública.
- Un sistema de firmas ciegas es posible $\Leftrightarrow$ $E(k, x)$ conmuta con $PR_A(x) \forall k$ es decir $\Leftrightarrow$ $\forall k PR_A(E(k, x)) = E(k, PR_A(x)) \Leftrightarrow E^{-1}(k, PR_A(E(k, x))) = PR_A(x)$.
- Es decir que si el comisario recibe un mensaje cifrado $E(k, x)$ y lo firma, por la identidad anterior el votante podrá recuperar $PR_A(x)$ y el comisario no habrá podido ver el voto ni inferir ningún tipo de información del mismo.
- De este modo si el votante envía $x||PR_A(x)$ de forma anónima se podrá verificar a posteriori que el voto x es válido y que ha sido firmado por el comisario.

Esto no es más que un método de ofuscación del voto, de modo que se puede firmar un voto sin ser capaz de acceder al contenido.

2.5. Proceso de Votado

En la época en la que vivimos en la cual la democracia es casi un estándar en las sociedades modernas el voto es una parte fundamental de ella, y como veremos en esta sección las garantías del voto en persona sobre papel son difíciles de implementar electrónicamente [25].

Los requisitos contemplados hoy día para el voto son los siguientes:

1. Solo pueden votar las personas particulares de un cierto área y nadie puede hacerlo si está fuera de la misma. Es decir, se ha de acotar de antemano quienes pueden votar e impedir que gente ajena a este conjunto pueda votar(empresas, comunidades autónomas, países, universidades).
2. Una persona puede votar un máximo de 1 vez. Esto implica que una persona no pueda falsear la votación aún perteneciendo al conjunto de personas autorizadas a votar.
3. No se puede permitir la suplantación de identidad.
4. El voto ha de ser secreto. Ya que el voto debe ser libre se debe garantizar que no se pueda saber de forma pública a no ser que el votante lo desee.
5. Los votos no han de ser manipulables ni modificables. Una vez que el voto se ha hecho se contabilizan los votos se debe asegurar que los votos son fieles a los de los votantes y que nadie ha hecho un pucherazo o una inserción/sustracción de votos a gusto.
6. El conteo de votos debe de ser preciso.
7. El conteo de votos debe de ser auditable.

En principio hay puntos de los requisitos que pueden parecer contradictorios como por ejemplo que un voto sea anónimo pero a la vez auditable. ¿Cómo podemos llegar a verificar que una persona ha votado y no tener ni idea de qué ha votado? En el caso del voto físico tenemos solución a ello, en el voto electrónico la solución no es trivial en absoluto, más adelante veremos un protocolo con el que se consigue.

2.5.1. Voto Físico

Antes de empezar a analizar el voto electrónico veamos el procedimiento del voto físico para unas elecciones y cómo garantiza estas condiciones para ver si pudiese ser similitudes criptográficas que emulasen el sistema de voto físico para ser al menos tan bueno como éste.

- Los ciudadanos están dados de alta en el censo y poseen todos un documento acreditativo único que les identifica. El requisito 1 se cumple gracias a este documento, que servirá más tarde para asegurarse de que votan solo determinadas personas (área nacional) y que la gente que no forme parte no pueda hacerlo (no tienen dni).

- Cada persona se acerca con un sobre cerrado con su voto introducido en él que trae desde casa o prepara en el colegio electoral (se establecen cabinas de voto para asegurarse de que el que no lo desee no sea visto) requisito 4.

- La mesa electoral verifica la identidad del ciudadano comparando su rostro con el del dni requisito. 3 Y además solo unas pocas personas pueden votar en una misma mesa electoral requisito 1.

- La mesa electoral verifica que esta persona está autorizada a votar en esa misma gracias a un documento emitido por la sede central en la que se les da a conocer las identidades de las personas que van a votar. Una persona solo puede votar en una única mesa determinada. Una vez hecho se hace notar en varias copias de un documento, garantizando así el requisito 2, Además satisface en parte el requisito 7 ya que a posteriori se puede verificar quién ha votado sin saber qué ha votado.

- Una vez termina el periodo de votos se procede al recuento de los mismos. Esto se hace por separado en paralelo por diversas mesas y por múltiples personas de probablemente con ideologías políticas diferentes, además de haber sido elegidas aleatoriamente y que no existan, en la gran mayoría de los casos, lazos de amistad, familia, o interés común entre ellos. Esto garantiza(en probabilidad) que no se pueda proceder a una manipulación de los votos ya que al ser de diferentes ideas políticas tendrían que ponerse de acuerdo primero en actuar en común de forma deshonrosa y en a que partido favorecer o perjudicar. Por lo que se asegura, en la medida de lo posible, el requisito 5 y el requisito 6.

- Acto seguido se transmite de forma telemática el conteo de votos para conocer los resultados de forma rápida, y se mandan los votos sellados para el recuento en la organización requisito 7.

2.5.2. Voto Electrónico

A la hora de mejorar la experiencia del voto de los votantes y adaptarse a las nuevas tecnologías hay pasos intermedios hasta la implementación del voto electrónico, a destacar el OMR(Optical Marks Recognition), DRE(Direct Register Electronic), MVS(Mail Voting System), RVS(Remote Voting Systems) [26].

OMR: Estos sistemas emplean el voto físico en el que la papeleta de votado posee unas marcas en las que se indica el voto y una máquina puede hacer la lectura del voto sobre la papeleta, esto acelera mucho el proceso de conteo sobre el voto físico tradicional.

DRE: Estos sistemas hacen uso de equipos electrónicos para recoger el voto, sea por ejemplo una cabina de votado en la que una persona pulsa el botón correspondiente a su voto, esto es una avance respecto a OMR ya que el registro, como su nombre indica, del voto se hace directo y no hace falta recogerlo para un conteo posterior.

MVS: Los sistemas MVS hacen uso de una autenticación previa para poder emitir el voto por correo, mediante un sobre sin remitente a la administración de voto, para que el voto sea contado posteriormente, la ventaja evidente de este sistema es que la gente con movilidad reducida o con dificultades para acudir a un centro de voto puedan ejercer su derecho a voto.

RVS: Los sistemas de voto a distancia permiten la autenticación y emisión del voto mediante una plataforma electrónica en la que se hace de forma instantánea y puede presentar métodos de auditoría reproducibles. Sin embargo el voto electrónico posee significativas diferencias con el voto físico [25]

Ventajas del Voto Electrónico: El voto electrónico supone un problema complejo que resolver, pero ¿Qué obtenemos gracias a la implementación del voto electrónico? [27] Entre muchas, las siguientes características:

1. Rapidez en el escrutinio de votos.
2. Accesibilidad al voto para gente con dificultad de acceso a los centros de votado.
3. Prevención de errores, ya que los sistemas de voto electrónico si tienen errores son aislados y no se anulan urnas, por ejemplo.
4. Posibilidad de auditoría por parte del votante.
5. Menor necesidad de recursos y de infraestructura ya que no hay que establecer mesas de votado ni comisarios.
6. Conveniencia para el votante, ya que no se tiene que desplazar.
7. Mayores horarios de votado gracias a que se necesitan servidores levantados y no mesas con personas.
8. Mayor número de elecciones dado la facilidad de realizar unas una vez la ifraestructura esté montada. Como por ejemplo Suiza, que se habla de su democracia directa gracias a la cantidad de referendums que se hacen para conocer la opinión de los ciudadanos y actuar en consecuencia.

Durante el proceso de votado se suceden las siguientes fases:

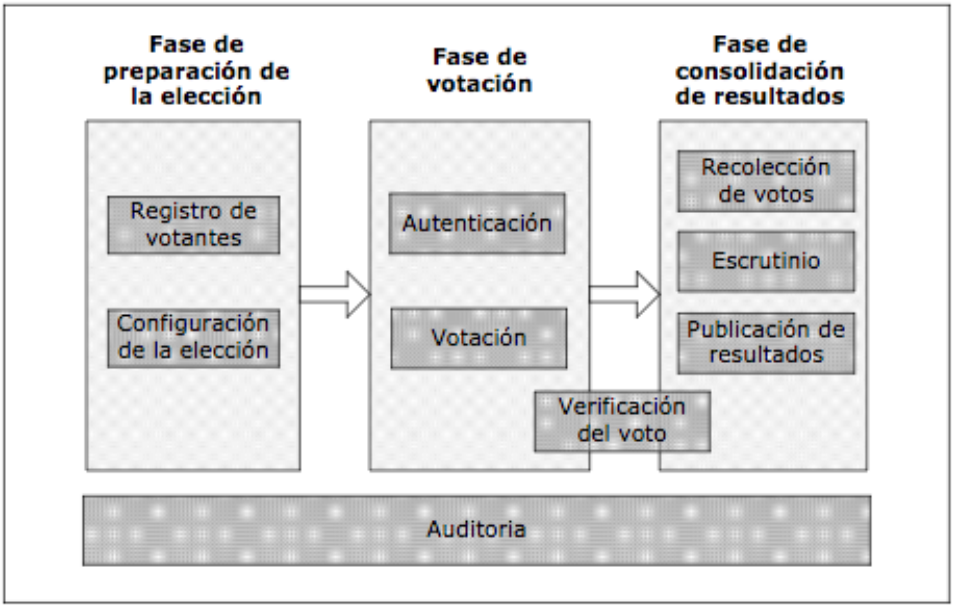

Figura 2.1: Proceso de Votado.(Fuente: *www.wikipedia.org*)

El votado electrónico simplifica significativamente los procesos de la fase de votación y de la fase de consolidación y escrutinio de votos. Sin embargo se complica más la fase de registro de votantes.

La fase de registro es un problema que corresponde a la identidad digital. Gracias a que este proyecto está enfocado a universidades y pequeñas empresas vamos a suponer esta fase ya hecha y resuelta, ya que en este tipo de organizaciones todos los miembros poseen una identidad digital, véase por ejemplo su correo institucional o su certificado digital de la organización. Estas organizaciones cuentan con un proceso de registro en el cual se verifica la identidad del miembro. Véase su contrato laboral o su matrícula universitaria. Un protocolo de registro seguro mediante el uso del correo electrónico es [28].

En este proyecto, al asumir que los votantes tienen un correo institucional se puede realizar el registro en la votación de forma sencilla ya que se asume que solo el votante tiene acceso a este correo y por tanto se le pueden transmitir los detalles de voto y las credenciales correspondientes a través de este correo.

Es por esto que en este proyecto se abarca el problema de la fase de votación y consolidación y escrutinio de votos suponiendo el registro de votantes hecho como se refleja en la figura 2.1. Sin embargo solucionar el problema del voto electrónico es complejo, veamos una serie de nociones que nos van a ayudar a comprender por qué solucionar el problema del voto electrónico está lejos de ser trivial.

Lo primero de todo es darse cuenta de que el requisito 1 del voto, requiere tener un control de la población que vota. Esto es lógico pues en una votación no es deseable que cualquiera sin identificación pueda votar, por lo que una persona a la hora de votar ha de ser identificada y autorizada para ello. Un modo de identificar a las personas criptográficamente es mediante certificados digitales, y es de los más utilizados. Y aquí reside el problema principal de la criptografía moderna convencional. ¿Cómo podemos conseguir que una persona identificada mediante su clave pública pueda votar anónimamente? Esto puede resultar contradictorio, y de hecho con la criptografía moderna convencional podría llegar a realizarse, eso si, con ciertas limitaciones en cuanto a la auditoría del voto se refiere.

Una noción de por qué es tan complicado realizar el voto electrónico es la siguiente, si suponemos que una persona pide autorización para votar mediante su certificado digital, esta autorización tiene que ser entregada por la autoridad que identifique al votante. Es por esto que la autoridad que autoriza al voto a un votante no puede ser la misma que reciba el voto. Uno de los mayores problemas del voto es entonces como ser capaces de recibir un voto, sabiendo que viene de un votante válido, sin poder averiguar nada del votante. Esto veremos como se resuelve más adelante.

Vulnerabilidades Típicas de un Sistema de Voto Electrónico: Debido a la naturaleza de las votaciones son muy susceptibles de ser atacadas, aquí recogemos las vulnerabilidades a priori de cualquier sistema de votado electrónico [7] en las cuales una implementación deficiente puede llevar a la completa vulnerabilidad de las elecciones.

1. **Deficiente Registro de Votantes:** Si la fase en la que el censo realizado para determinar que subconjunto de personas son las autorizadas a votar se podría manipular para introducir votantes no legítimos.
2. **Deficiente Uso de Recursos Criptográficos:** La criptografía conforma una parte clave en el votado electrónico ya que va a determinar como de cierto es el anonimato del votante o la integridad de la votación, esta vulnerabilidad puede ser explotada por muchas razones, como puede ser el uso de criptografía obsoleta como puede ser DES, o funciones Hash con colisiones, o utilizar criptografía no obsoleta pero definida mediante unos tamaños de clave suficientemente pequeños como para poder atacar el algoritmo.
3. **Proceso de Autenticación Débil:** Esto se da cuando no se consigue la siguiente condición: Un votante se autentica en el sistema de votado $\Leftrightarrow$ está autorizado a votar. De este modo un atacante podría intentar manipular las elecciones o introducir votos ilegítimos.
4. **Deficiente Control de Acceso:** Elementos tales como ficheros, logs del sistema, bases de datos, deben ser debidamente protegidos de accesos ilegítimos, si no se hiciese así un atacante podría hacer un uso indebido de estos recursos.
5. **Canales de Comunicación Inseguros:** Dado que el voto electrónico tiene como parte esencial transmitir el voto a través de la red, un uso de un canal inseguro comprometería la seguridad del voto del votante.
6. **Deficiente Registro de logs:** Un deficiente registro de la actividad de logs de un sistema puede derivar en no detectar intentos de ataque o de manipulación del sistema de voto.
7. **Procesos Deficientes de Verificación:** Durante un protocolo de votación electrónica se tienen que realizar diferentes procesos de veirificación para asegurar la validez del voto y la legitimidad del votante, de no realizar estas verificaciones se podrían introducir votos ilegítimos o manipular los existentes sin comprobar la correcta configuración.

Posibles Ataques a un Sistema de Voto Electrónico: Los ataques más relevantes a un sistema de votación electrónica son los siguientes [29]:

1. **Suplantación de Identidad en el Proceso de Registro:** Si el registro se realiza de forma online y alguien consigue impersonar a otro votante potencialmente podría emitir un voto de forma correcta con la identidad de otra persona.
2. **Manipulación del Censo Electoral:** Antes de realizar una votación se debe esclarecer cual es el subconjunto de personas autorizadas a votar, si se consiguiese manipular se podrían introducir votantes ilegítimos en las votaciones o desautorizar a votantes legítimos.

3. **Manipulación del Software de Voto:** Si el software utilizado contiene algún tipo de información relevante el votante podría manipularla para su propio beneficio en las votaciones.

4. **Daño del Hardware de Votado o de la Red:** Dado que estos votos tienen que ser transmitidos por internet, y potencialmente existen equipos de votado electrónico establecidos durante las votaciones, se pueden convertir en los objetivos por parte de atacantes que tengan como intención sabotear las elecciones.

5. **Configuración Errónea de la Votación:** Si el protocolo de votado tiene unos parámetros erróneos la votación no se va a poder realizar de forma satisfactoria.

6. **Voto Múltiple:** Esto sucede cuando el proceso de autenticación permite a un mismo usuario insertar votos en el sistema múltiples veces, dando lugar a una discrepancia entre número de votantes y número de votos.

7. **Inyección de Votos Ilegítimos:** El objetivo de este ataque es manipular el resultado de las elecciones, se hace alterando la base de datos donde se almacenan, si estos no poseen un sistema de verificación apropiado.

8. **Sustitución de Votos:** El objetivo de este ataque es el mismo que el anterior, pero en este caso se evita la discrepancia entre número de votos y número de votantes.

9. **Captura de Votos:** La captura de los votos supone un problema respecto a la privacidad del votante, ya que podría intentar revelar la identidad del votante rompiendo su anonimato o bien realizar un ataque Man in the Middle (Ver Anexo M) para manipular el voto del votante.

10. **Denegación de Servicio:** Evitar que ciertos usuarios, o todos, no tengan acceso al sistema de votado puede resultar una situación atractiva para un atacante, esto puede resultar en un ataque que inhabilite alguno de los procesos de votado o todos mediante un ataque DoS/DDoS (Ver Anexo L).

11. **Confabulación:** En una mesa electoral existen distintos miembros con distintos intereses para evitar que se puedan aliar para manipular las elecciones, sin embargo, de ponerse de acuerdo, suponen un riesgo a la integridad y veracidad de los resultados de las elecciones.

12. **Coerción:** Esto supone un problema en todos los sistemas de votado actuales, pero especialmente al electrónico, el riesgo de venta de votos o la obligación a alguien de votar de determinada forma es más grande en el voto electrónico, ya que en estos sistemas no se tienen cuerpos de seguridad vigilando que el voto se ejerce de manera libre, ya que cada uno puede votar desde sitios potencialmente muy diferentes.

Dada la naturaleza del voto electrónico la seguridad es de carácter esencial y no accesorio, por ello será uno de los aspectos que más se van a tener en cuenta a la hora de desarrollar el protocolo.

Sistemas de Voto Electrónico Actuales: Actualmente existen muchas alternativas a la hora de realizar un voto electrónico, con diversas implementaciones y múltiples tecnologías involucradas. Hoy día nos encontramos, por ejemplo, con sistemas basados en firmas ciegas, las cuales se han explicado con brevedad y se otorga un breve análisis de un protocolo diseñado en el Anexo J.

Nos encontramos también con protocolos basados en Mix-Networks [30], las cuales se basan en tener una lista de votos, y cuando un nuevo votante quiere introducir su voto, toma la lista,

introduce su voto, y mezcla la lista, de este modo, entrega la nueva lista de votos adjuntando la prueba criptográfica de que la mezcla de votos se ha efectuado de forma correcta. Es un esquema muy interesante desde el punto de vista del anonimato, una explicación más en detalle se puede encontrar [5].

Otro de los sistemas utilizados para el voto electrónico es basado en contadores criptográficos, como vienen explicados en [4] que se basan en la capacidad por parte de miembros autorizados, en este caso votantes, de incrementar o decrementar un valor, para ellos incierto. En este caso los contadores sería el número de votos para cada candidato. Estos contadores criptográficos funcionan de modo que no se obtiene información de los votantes a partir del voto ya que aumentan el valor del contador de forma anónima y estos no conocen los resultados hasta que el administrador los publique.

También existen protocolos basados en firmas grupales como los reflejados en [31] [32] y [33], sin embargo estos últimos son muy recientes y es por esto por lo que este proyecto busca solucionar el problema del voto electrónico de forma independiente a los trabajos que hay hoy día usando firmas grupales.

Hay esquemas para un acceso más simplificado y para votaciones de menor escala, como por ejemplo Agora Voting(*https://agoravoting.com/*). Sin embargo este protocolo se ha visto inmiscuido en polémicas por la falta de validez y posibilidad de voto múltiple.

Son muchos los países que ya utilizan en cierto grado un esquema de voto electrónico, siendo este una adición al voto físico. Como por ejemplo Alemania, Brasil o Suiza. Este último ha estandarizado las votaciones, las cuales ya se hacen de forma muy habitual dada la agilidad de las votaciones. Es por esto que este tipo de sistemas presentan una alternativa al concepto de democracia indirecta la cual practicamos la mayoría de países, y se puede pasar a un sistema híbrido en el que la democracia directa aparece de forma frecuente. Esta consecuencia no viene garantizada con el voto electrónico, pero sin embargo es un gran aliciente a cambiar el paradigma democrático y por tanto es un problema muy interesante de estudiar.

3

Sistema, diseño y desarrollo

3.1. Protocolo de Voto Electrónico basado en firmas grupales

Como se ha comentado anteriormente el primer protocolo estudiado fue el de firmas ciegas, el cual se encuentra en el Anexo J. Como se ha comentado después de estudiar el de firmas grupales se eligió este como protocolo principal de estudio de este trabajo.

En esta sección se describirá y analizará el protocolo diseñado para realizar un voto electrónico mediante el uso de las firmas grupales. El análisis incluye una descripción del diseño, de la seguridad, y del anonimato que aporta.

Resumen del Protocolo Este protocolo viene explicado en la figura 3.1. A continuación se explicará a modo informal el protocolo y más adelante a nivel técnico explicando en detalle todas las características del mismo.

Este protocolo de voto electrónico está basado en un grupo de cifrado revocable, que estará en un servidor que además entregará pseudónimos, al que llamaremos Server G1. Un servidor de entrega de Tokens de votado TokenV que llamaremos TokenV Dispatcher. Y un servidor de votado al que llamaremos Voting Server.

Para que este esquema ofrezca garantías de protección ante la organización los servidores Server G1, TokenV Dispatcher y Voting Server deben ser independientes y este sistema será seguro siempre que estos organismos no colaboren, a no ser que se les exija, como puede ser bajo una auditoría del proceso de votado. El esquema de firmas grupales utilizado en este protocolo ha sido CPY06(Ver 2.4.2), que ha resultado ser uno de los más eficientes de la librería libgroupsig, como bien viene indicado en el artículo [1], cuyas gráficas las encontramos en el Anexo Ñ.

Idea del Protocolo Como veremos más adelante, este esquema se basa en la idea de que los votantes utilicen su identidad para acto seguido anonimizarse, esto podría verse como si un votante fuese a una organización en la cual, pueden verificar quien es, por ejemplo, mediante su dni y su rostro, y que esta organización le de un modo de votar sin ser reconocido. A modo ilustrativo vamos a explicar una analogía de lo que haría este protocolo con un votante físico con este esquema.

1. El votante va a la administración de anonimato, en esta el votante muestra su DNI y su rostro, la administración verifica que es un votante autorizado y acto seguido le entrega una máscara que le cubre completamente la cara. La organización saca una foto al dni de la persona junto con la máscara y la guarda para más tarde. Esto se corresponderá con la interacción con el servidor G1 en la que el cliente se identifica como votante válido y el servidor le da un pseudónimo, que como veremos se llamará TokenPS.

2. El votante a la hora de recoger su sobre de voto en la otra organización independiente, se pone esta máscara, de este modo la organización que entrega el sobre no sabe quien es, pero quieren verificar que el votante está autorizado para votar, así que envían un mensaje a la organización dispensadora de máscaras con una foto del votante(con la máscara puesta). La dispensadora de máscaras reconoce la máscara y responde que efectivamente ellos han dado esa máscara a alguien, sin decir a quien, a pesar de saberlo. Acto seguido le dan un sobre válido con un identificador al votante. Y guardan una foto del sobre junto con la máscara. Esto se corresponde con la obtención de token de votado, que se llamará TokenV. De este modo el votante obtiene su autorización anónima y correcta de votado.

3. El votante envía por correo anónimo su voto en el sobre identificado para ello, de este modo la organización sabe que el voto es correcto pues el sobre está identificado para ello. Esta parte se corresponde con la última parte del protocolo que es la de la entrega anónima del voto.

Este sistema hace uso del anonimato mediante pseudonímia, un usuario cuando va a por su sobre está identificado por su máscara y el que le da el sobre no puede averiguar la identidad del votante. Además existe un modo de abrir la votación y auditarlo. Si la entidad dispensadora de máscaras se pone de acuerdo con la dispensadora de sobres pueden poner sus fotos juntas y averiguar qué votante ha votado qué. Esto puede sonar no deseable, pero es un requisito fundamental en los sistemas de voto electrónico, como ya veremos más adelante, para realizar una auditoría del proceso de votado. Además en este caso la apertura del sistema de votado es de secreto compartido, es decir requiere de la colaboración de las dos entidades para poder abrirla, lo cual es deseable para evitar que un mal administrador comprometa el sistema.

Requisitos del sistema En esta sección estudiaremos cuales son los requisitos iniciales que se contemplan en este esquema para su correcto funcionamiento, entendiendo también por correcto, seguro y privado.

1. Cada usuario conoce su clave de miembro de grupo de antemano, es decir ha sido transmitida de forma segura. Y por lo tanto puede firmar como miembro válido sin revelar su identidad.

2. Se asume que la identidad de un usuario es equivalente a su clave de miembro de grupo.

3. Cada servidor posee una clave pública y una clave privada con su correspondiente certificado digital emitido por una autoridad en la que confían que tanto los votantes como los otros equipos.

4. Los organismos del servidor G1 ,del servidor TokenV Dispatcher, y el servidor de voto son completamente independientes y fiables (Ver figura 3.1).

5. Todos los miembros autorizados a votar en el sistema pertenecen a un grupo de cifrado y solo ellos poseen una clave de miembro

6. La clave del grupo de miembros autorizados a votar es pública por tanto se puede comprobar que un mensaje firmado grupalmente efectivamente viene del grupo

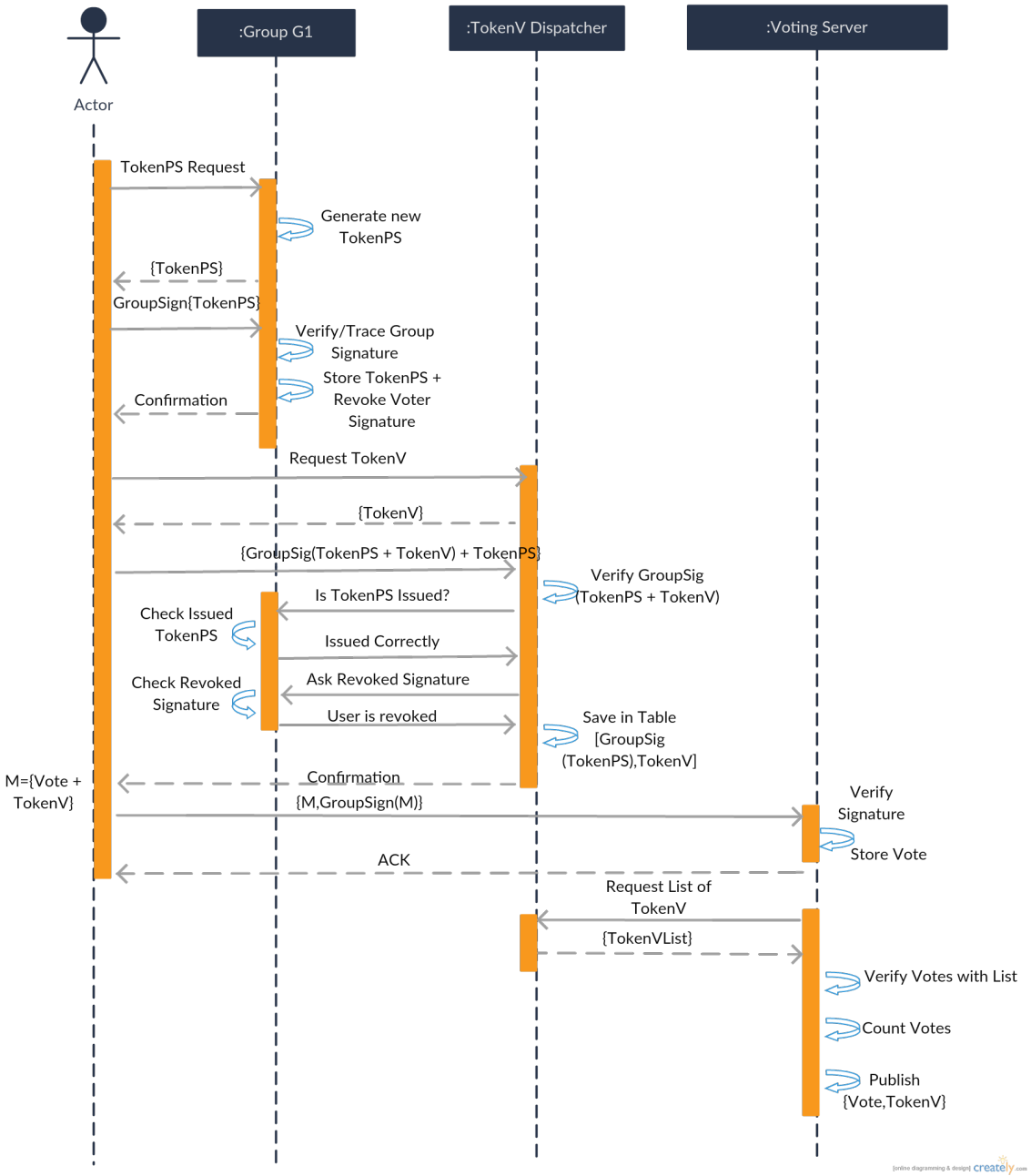

Figura 3.1: Protocolo de Voto Electrónico usando Firmas Grupales

Diseño y Descripción del Funcionamiento del Protocolo de Votado Electrónico.

1. El votante se conecta al servidor G1 con TLS/SSL (Ver Anexo D) para que la comunicación sea segura y pide un Token Pseudónimo (TokenPS).

2. El servidor G1 genera un Token Pseudónimo aleatorio nuevo y temporal al usuario para que lo firme y demuestre pertenencia al grupo.

3. El votante envía la firma grupal de este Token Pseudónimo (TokenPS) emitido.

4. El servidor comprueba primero que efectivamente es una firma de grupo válida(esta acción no abre la identidad del votante), y posteriormente que este usuario no está revocado mediante la trazabilidad de la firma. Es decir, se comprueba que se le puede asignar este

TokenPS. En caso de que se compruebe que este votante ya posee un TokenPS se le comunica y el TokenPS que se le ha enviado no se activa, por lo que ese TokenPS no podrá ser usado para votar.

5. El servidor revoca a este votante para que no pueda obtener mas Token Pseudónimos. Y Acto seguido guarda en una tabla interna privada [TokenPS,groupsign(TokenPS),identidad votante, Timestamp]. Esta base de datos puede estar cifrada para una mayor seguridad.
6. El votante se desconecta del grupo G1.
7. El votante va con su TokenPS al servidor TokenV Dispatcher y solicita Token de votado mediante TLS/SSL.
8. El servidor TokenV Dispatcher genera un TokenV temporal y solicita al usuario cuál es su TokenPS y una demostración de que es un usuario de votado válido mediante la firma grupal del TokenPS y TokenV.
9. El votante genera la firma grupal de TokenPS junto con TokenV y se lo envía a A junto con su TokenPS.
10. TokenV Dispatcher verifica la firma grupal, si pasa el test entonces pregunta a G1 mediante TLS/SSL si la firma que se adjunta ha sido revocada(viene de un usuario que ha recibido ya un TokenPS) y si el TokenPS es válido(es decir no es una cadena aleatoria generada por un votante fraudulento).
11. G1 mira en la tabla y verifica que ese TokenPS está emitido. Y si esa firma viene de un miembro de grupo revocado. Esta respuesta es binaria por lo que el servidor A no obtiene información del usuario que quiere votar.
12. TokenV Dispatcher comunica activación de TokenV al votante. Acto seguido marca que a ese TokenPS no se le deben otorgar mas TokenV mediante [TokenPS, TokenV, GroupSign(TokenPS+TokenV), Timestamp]. Esta tabla es privada y solo la conoce TokenV Dispatcher.
13. El usuario ya con este TokenV, que es secreto, envía de forma anónima "Voto + TokenV + la firma grupal de todo lo anterior.ª Voting Server, todo usando TLS/SSL para que la comunicación sea segura.
14. El servidor Voting Server comprueba que la firma es válida lo primero, para verificar que es un miembro del grupo de votado válido, acto seguido comprueba que ese TokenV no ha votado aún, y de ser así guarda el voto de modo correcto.
15. Se espera un tiempo hasta que acabe la jornada electoral.
16. El servidor de conteo de voto pide a TokenV Dispatcher la lista de TokenV autorizados a votar, y usa esta lista para implementar el conteo de votos.
17. El conteo de votos consta de los siguientes pasos: El servidor de conteo comprueba que los votos no han sido alterados ni modificados mediante la firma de cada uno de ellos, acto seguido busca el TokenV correspondiente a cada voto en la lista que TokenV Dispatcher le ha transmitido y si es así cuenta el voto.
18. Cada usuario puede comprobar que su voto ha sido contado mediante su TokenV que solo el conoce, estos resultados se publicarán en un servidor independiente. Este TokenV se utiliza como resguardo, pues cuando se publiquen los votos irán acompañados de este TokenV, y ni G1 ni Tokenv Dispatcher saben a que votante pertenece. De cualquier modo y de modo independiente el usuario puede hacer una auditoría completa de su voto, que se explicará más adelante.

¿Qué hemos ganado utilizando firmas grupales? Es una pregunta importante que plantearse pues las firmas grupales son un proceso complejo que, si pudiésemos eliminar del sistema obteniendo un sistema equivalente, sería absurdo usarlas. Bien, veamos que, el uso de las firmas grupales está justificado:

- Al usar firmas grupales permitimos que una persona pueda demostrar que forma parte del grupo sin necesidad de revelar su identidad, como podemos ver en el esquema, el servidor TokenV Dispatcher puede verificar que el usuario en cuestión es un miembro válido del grupo, así pues antes de comunicarse con el servidor G1, puede determinar si el mensaje recibido es correcto por parte del cliente o no. Otra parte en la que esto es tremendamente relevante es en la de la entrega del voto, en esta parte se entrega el voto de forma anónima, esto si no se aplica ninguna de control permite que usuarios anónimos envíen votos falsos solo para que sean almacenados en el servidor y retardar el proceso de votado. Al utilizar firmas grupales, al no tener la clave de administrador el servidor TokenV Dispatcher no puede averiguar la identidad del votante que ha emitido la firma. Y gracias a esta medida evitamos este tipo de ataque.
- Gracias a la revocación podemos llevar un control de qué usuario ha votado ya impidiendo múltiples votos.
- La capacidad de auditoría individual que tiene cada votante sobre el proceso, gracias a las firmas grupales se pueden evitar "pucherazos", para empezar porque solo los miembros del sistema están autorizados a votar, y esto es, criptográficamente, son los únicos capaces de emitir votos que acepte el sistema. Y lo probablemente más importante, pueden aportar demostraciones de que son los dueños de un TokenPS, de un TokenV y de un voto. Son capaces de emitir pruebas criptográficamente no falseables sobre la propiedad legítima de estos Tokens y su voto. Hoy día los sistemas de votado electrónico generalmente garantizan un control no completo del voto, es decir, les otorgan una cadena identificativa con su voto, y pueden ver que su voto efectivamente está publicado, con las firmas grupales vamos más allá, un usuario es capaz de emitir una prueba criptográfica de que es dueño de un TokenPS en particular, y nadie más puede emitir esta prueba.
- La capacidad de realizar una auditoría global del proceso de votado, gracias a la división de secreto.

3.1.1. Análisis de Funcionamiento Correcto del Protocolo

En esta sección estudiaremos comoel por qué el protocolo funciona de modo correcto y cumple con los requisitos necesarios del voto.

1. Solo pueden votar los votantes autorizados para ello, pues en el grupo G1 se comprueba la identidad de la persona y la pertenencia al grupo.
2. Una persona solo puede votar una vez, pues si una persona vota 2 veces o bien ha conseguido 2 pseudónimos, lo cual es imposible ya que hay un control y a cada persona se le emite un máximo de 1 pseudónimo para votar, o bien consigue 2 tokens de votacion, pues para ello necesitaría 2 tokens pseudónimo lo cual ya hemos dicho que es imposible, o que el servidor de tokens de voto emita 2 tokens de voto al mismo pseudonimo lo cual es imposible pues marca a que tokens pseudonimo se les ha emitido token voto.
3. La identidad de una persona no puede ser suplantada ya que la identificación es unívoca mediante su certificado digital en G1.

4. El voto es secreto, veamos por qué. El voto consta de el voto como tal y el token de votado emitido por TokenV Dispatcher. Esto se publica y todos lo pueden ver, veamos, el token de voto solo lo conoce TokenV Dispatcher y el votante. Puede TokenV Dispatcher comprometer la identidad del votante mediante el token? no, no puede, ¿Por qué? porque TokenV Dispatcher solo conoce a que pseudónimo se le ha otorgado un token de voto. El único que podría conocer la identidad del pseudónimo es G1, dado que hemos asumido que G1 y TokenV Dispatcher son organismos independientes no colaborativos el anonimato está garantizado.

5. El voto no ha de ser manipulable ni modificable, y en este caso no lo es, veamos por qué. Al llegar el mensaje de voto, se comprueba la firma grupal en el servidor de conteo de voto, esta firma sirve como sistema de garantía de que no se modifica, ya que cualquier modificación produciría una firma grupal no válida.

6. El conteo de votos tiene que ser preciso, y este lo es, ya que no hay nada que calcule mejor que un ordenador.

7. El proceso debe poder ser auditable, bajo demanda y puesta de acuerdo entre todos los servidores de voto se podría poder llegar a auditar todo el proceso de voto de modo reproducible.

Ahora queda otra cuestión por afrontar que es el explicar lo siguiente: ¿Por qué utilizar 3 servidores? ¿Es necesario? La respuesta es que si (teniendo en cuenta que utilizamos firmas grupales, con otros esquemas utilizar menos servidores puede ser perfectamente viable). Panagiotis Grotas [29] recoge y analiza múltiples protocolos basados en varios servidores computacionales independientes.

Si utilizásemos solo un sistema, por el requisito fundamental del voto de que solo pueden votar las personas autorizadas a ello, este servidor debe conocer la identidad de un votante al menos al comenzar la conexión. Y con firmas grupales esto implica que el servidor conocería de inmediato su identidad ya que utilizamos un grupo revocable, con un solo servidor el usuario al autenticarse si vota, no tiene anonimato, y si el servidor le diese un TokenV, y este usuario votase más tarde seguiría no habiendo anonimato ya que este TokenV ha sido generado por el mismo servidor que lo recibe.

Es por esto por lo que surge la idea de utilizar un servidor para la autenticación y otro para el voto. Supongamos que utilizamos este esquema, usando tan solo dos servidores, para que el servidor que da identidades pueda estar seguro de que el segundo servidor no introduce votos a placer. A la hora de publicar resultados se tiene que adjuntar alguna prueba además del voto que alguien pueda confirmar que efectivamente es un voto correcto, y en este caso la única prueba posible a usar es la firma grupal, resultado se rompe el anonimato si se adjunta esa prueba en el voto.

Lo mismo pasaría si utilizásemos el primer servidor para autenticar y diese un TokenV, el segundo servidor tiene que adjuntar una prueba de voto que satisfaga tanto a votantes como al primer servidor, y en este caso la única prueba viable es la firma grupal, la cual permitiría al primer servidor romper el anonimato de todos los votantes. Si utilizamos el esquema anterior basado en pseudonimia y un tercer servidor, obviamente el segundo ya no podrá falsificar votos, pues bajo auditoría si ha introducido TokenV a placer se podrá verificar las firmas grupales guardadas y comprobar que se han introducido de modo incorrecto. De este modo adjuntando el TokenV como prueba pública de Voto, el servidor de voto no podrá introducir votos a placer pues el servidor TokenV Dispatcher podrá detectar el fraude, y del mismo modo que en el ejemplo anterior, si es TokenV el que introduce votos con la auditoría explicada anteriormente se detectaría el fraude.

3.1.2. Análisis de Seguridad del Esquema

Una vez queda establecido el protocolo debemos verificar que el protocolo es seguro, en esta sección se explicará con más detalle de qué modo se harán estas conexiones. Como se asegura el anonimato en las partes que se necesite, etcétera. Se analizarán las vulnerabilidades conocidas, las posibles consecuencias de corrupción y posteriormente un análisis de anonimato del protocolo [26].

Una de las claves de este protocolo es asumir que las personas ya han recibido su clave de miembro de grupo, dado que este esquema está pensado para funcionar y ser eficiente en entornos de población moderada (Universidades, instituciones y empresas) es legítimo asumir que los votantes están identificados por un correo electrónico al que el servidor G1 puede enviar la clave una vez generadas, o si los votantes tienen un certificado digital pueden elegirlas en un periodo pre-electoral en el que se pueden hacer con una clave solo los usuarios autorizados a votar. Sea del modo que fuese, en este esquema se supone que los votantes ya han recibido de forma correcta y segura su clave de miembros de grupo.

En todas las comunicaciones que se hagan de forma anónima a cualquier servidor se utilizará la clave pública de este servidor mediante TLS/SSL para asegurarse que el mensaje solo puede ser leído por el servidor destino y otorgar un test que asegure la integridad del mensaje en esta comunicación, como viene explicado en el anexo TLS/SSL garantizamos que esto funcione de forma correcta y la comunicación TLS/SSL es suficientemente segura. Además esta tecnología protege de ataques como packet replay, y protege la conexión como se explica en el Anexo D.

En este esquema están separados el servidor de dispensador de pseudónimos (G1) y el servidor dispensador de tokens de votado (TokenV Dispatcher), ¿Por qué? La razón es muy simple, al final tiene que haber un servidor en el que se vote, y este servidor tiene que recibir los votos de las personas autorizadas, por ejemplo por un Token. Ahora bien, este Token ha de ser dispensado solo a los miembros autorizados a votar, por lo que es imprescindible que en algún momento los usuarios muestren su identidad. Si es el servidor dispensador de Tokens el que asigna estos Tokens a los usuarios, el servidor puede saber al final quién ha votado qué. Es por esto que si lo que recibe el dispensador de TokenV, el dispensador de TokenV no puede averiguar la procedencia de un voto, lo máximo que puede llegar a conseguir es qué pseudónimos ha votado qué, lo cual protege la identidad del votante.

En el voto electrónico en general uno de los mayores problemas es garantizar el anonimato y a la vez establecer un control sobre los votantes de modo que no sea posible realizar varios votos con la misma identidad. En cualquier esquema de este tipo, el paralelismo puede suponer un problema, este problema surge si no se posee de un adecuado control de concurrencia, es decir, que si no se tienen en cuenta ciertos factores podríamos tener un sistema vulnerable a las múltiples identidades, veamos un ejemplo de lo siguiente:

- Un votante tiene ya su Token Pseudónimo y pide de forma paralela, digamos que en N conexiones diferentes al mismo tiempo, sus correspondientes N tokens de votado.
- En cada uno de los mensajes el votante fraudulento envía el mismo token pseudónimo y la correspondiente firma grupal del token.
- A recibe estos N mensajes y hace las correspondientes N preguntas al servidor G1 y en todas las conexiones el servidor G1 le responde a A que la firma grupal efectivamente está revocada y que el token pseudónimo se ha emitido de forma correcta.
- El servidor A lee en la tabla de Token V que tiene para cada una de las conexiones si a ese votante⇔Groupsig(TokenPS) se le ha emitido Token V.

- Supongamos que el control sobre este fichero se hace de la siguiente forma, se permiten múltiples lecturas a la vez pero solo una escritura a la vez y mientras se escribe no se puede leer. Entonces es viable que cada hilo de cada una de las N conexiones paralelas lea al mismo tiempo que efectivamente a ese votante no se le ha asignado TokenV.

- Una vez todas las conexiones han verificado este suceso cada hilo designado a atender cada una de estas conexiones genera un TokenV y se lo envía al usuario, resultando en que un votante podría votar N veces siendo N un número deseado > 1. Por lo que supone un agujero tremendo de seguridad.

Como vemos un mal control de flujo sobre los datos guardados por los servidores puede suponer un agujero de seguridad en el mismo. Por lo que debemos asegurarnos en bloquear los recursos antes de escribir, y que el proceso de lectura sea bloqueante del mismo modo que lo es la escritura, de este modo si lo que hacemos es hacer un bloqueo, leer, verificar que no se le ha emitido tokenV, escribir, y acto seguido liberar el recurso solucionamos este problema. Es decir, en este caso, la primera conexión que vaya a dar el TokenV al usuario, bloquearía primero el fichero. Así el resto de conexiones estarían bloqueadas antes de leer hasta que esta primera terminase. Esta primera conexión primero lee en el fichero si a este pseudónimo se le ha asignado un Token de Voto. Comprobaría que aún no se le ha emitido uno, generaría un Token de Voto, lo guardaría, liberaría el fichero, y enviaría este Token de Voto al votante. Acto seguido entra alguna de las conexiones que estaba esperando a que el fichero estuviese disponible, y leería en la tabla que a ese pseudónimo ya se le ha asignado un Token de Voto. Notificándole al votante en cuestión el error y así asegurándonos de que a cada pseudónimo se le asigna como mucho un Token de Voto.

Del mismo modo, si el servidor G1 no lleva un control apropiado sobre las operaciones sobre el grupo, podría asignarle N Tokens Pseudónimo a un mismo usuario. ¿Cómo se podría solucionar esto de forma sencilla? Haciendo que la comprobación de que un votante está revocado y la revocación posterior estén ambas protegidas por el mismo proceso. Equivalente al concepto de transacción en una base de datos, una operación o se realiza de forma completa, o no se realiza nada.

Pérdidas de la conexión Una consideración importante a tener en cuenta es, ¿Qué pasa si se pierde la comunicación en medio del proceso de votado? ¿Qué consecuencias tiene? ¿Puede el votante en cuestión seguir votando? En la siguiente sección explicaremos que consecuencias puede tener perder la conexión en cualquier momento de la comunicación. Y como procederia el protocolo para permitir al votante seguir votando. Esto es muy importante ya que uno de los ataques mas comunes en las comunicaciones es el ataque DoS(Denial of Service) o DDoS(Distributed Denial of Service). Esto es dado a que el DoS generalmente no necesita saber ningun tipo de informacion de cifrado ni de los secretos compartidos por los miembros de la comunicacion ni de criptografia si quiera, el unico objetivo en este caso seria tirar las conexiones legitimas de voto para por ejemplo sabotear los resultados.

- Si el usuario pierde la conexión antes de que haya verificado la asignación del Token Pseudonimo, el servidor G1 no marcará el token Pseudonimo en cuestión como asignado y por tanto el cliente no podrá usar ese tokenPS para votar, pero como vemos por el proceso, al usuario no se le ha revocado aún del grupo de cifrado por lo que puede volver a conectarse para pedir otro TokenPS y de este modo proceder a votar.

- Si el usuario ha perdido la conexión antes de pedir Token de Voto pero teniendo Token Pseudónimo puede empezar la conexión de nuevo directamente hacia el servidor dispensador de TokenV ya que el servidor G1 tendrá marcado como que ese tokenPS está asignado.

- Si el usuario pierde la conexión con el servidor A antes de ser asignado el Token de Votado, puede reiniciar la conexión con el servidor A dado que la respuesta del servidor G1 al A no cambia si no se le ha asignado el TokenV.

- Por el contrario, si el usuario pierde la conexión una vez se le ha asignado TokenV pero no lo recibe, al reiniciar la conexión con el servidor, dado que el servidor guarda en una tabla [GroupSig(TokenPS),TokenV] A puede buscar GroupSig(TokenPS) en la tabla y si ya existe enviarle ese Token de Voto ya que GroupSig(TokenPS) es una forma univoca que hay de identificar a un votante valido.

- Si el usuario pierde la conexión antes de recibir el OK del servidor de voto, puede volver a enviar el voto y con este Token de Voto el servidor le puede comunicar que ya ha votado y el localizador de su voto, es importante ver aqui que un usuario recibe esta notificacion aunque envie un voto diferente al primer voto.

Vulnerabilidades Conocidas del Protocolo En esta sección explicaremos condiciones bajo las cuales el correcto funcionamiento del sistema puede verse comprometido y en cuanto afectarían estas vulnerabilidades al sistema, del mismo modo se establecen medidas preventivas para evitar que estas vulnerabilidades se materialicen.

Corrupción de la Administración G1: Esta condición es bastante elemental, pues la administración de la votación si esta corrupta puede manipular en gran medida los resultados veamos que podría hacer esta rama de la administración si tuviese malas intenciones.
Inyección de votos: Esta vulnerabilidad es bastante trivial, ya que en la vida real si la administración en la que votamos está corrupta, van a poder hacer con nuestro voto lo que les plazca. Ahora bien, si esta administración G1 estuviese corrupta, podría generar todos los Tokens Pseudónimo que le plazca y manipular los resultados de los votos como guste. Ya que cuando A le pregunte a G1 sobre los Token Pseudónimo G1 va a confirmar que son válidos. ¿Cómo podemos evitar esta vulnerabilidad en cierta medida? Al final G1 tiene que guardar en un registro a qué persona le ha emitido qué Token Pseudónimo, de modo que bajo auditoría G1 se vería comprometido, lo cual solventa esta vulnerabilidad bajo auditoría.
Impersonación de Votantes: Por lo explicado en el anterior apartado G1 solo podría intentar salirse con la suya utilizando entradas de miembros reales del grupo para poder inyectar votos y escapar impune de la auditoría. Supongamos que utiliza las identidades de las personas que no han votado hasta X minutos antes de que finalicen las elecciones. Entonces se podría hacer un análisis de carga de trabajo y el timestamp de los votos, lo cual haría sospechoso que un porcentaje significativo de votos se haga de modo tan seguido y una auditoría preguntando a los votantes de esta franja horaria revelaría las malas intenciones de G1. De este modo lo único que le queda hacer a G1 para insertar votos es utilizar identidades reales de los usuarios a lo largo del día. Lo cual es potencialmente detectable, ya que cuanto mayor sea la cantidad de votos que inserta G1, mayor es la probabilidad de que un votante tenga intención de votar y detecte que su voto ya ha sido usado. De este modo puede abrir una auditoría.
Denegación de Servicio: G1 podría negar entregar Token Pseudónimos a votantes o entregarlos y negarse a realizar el test que le pide A de forma correcta, de este modo podría limitar la gente que vota o invalidar las elecciones por completo. Si este servidor niega tantas conexiones se puede abrir una auditoría al respecto.

Corrupción de la administración TokenV Dispatcher: Del mismo modo que puede haber corrupción en el sistema G1 lo puede haber en el TokenV Dispatcher, repito que estas vulnerabilidades considerarlas es estudiar que pasa en el esquema si hay un error de base, pero que aún así merece la pena dejarlas documentadas.

Inyección de Votos: Si este servidor se pone a generar Tokens de Votado de forma indiscriminada, el servidor de votos será capaz de contarlos y de hecho los contará como válidos, sin embargo, hemos dicho que lo que TokenV Dispatcher guarda es [Groupsign(TokenPS),TokenV] De este modo la administración TokenV Dispatcher en colaboración con la auditoría puede verificar que Groupsign(TokenPS) pertenecen a usuarios revocados y cuales no, de hecho, puesto que TokenV Dispatcher no puede generar estas firmas, TokenV Dispatcher sería capaz de localizar que firmas Groupsign(TokenPS) han sido inventadas y por tanto no son legítimas y así localizar que TokenV han sido inventados por la administración TokenV Dispatcher.
Denegación de Servicio: Si la organización TokenV Dispatcher lo desease, podría rechazar ciertas conexiones, del mismo modo que en G1, se puede hacer un análisis de carga de trabajo y una auditoría de registros de red para ver una posible denegación de servicio por parte de la administración.

Corrupción del servidor de Conteo de Votos ServerV: Si este servidor es corrupto del mismo modo que los anteriores puede tratar de manipular los resultados de diversos modos.
Inyección de Votos: Si este servidor mete votos de forma indiscriminada una auditoría por parte de la administración TokenV Dispatcher puede determinar que votos ha metido ya que la lista auténtica de TokenV la tiene TokenV Dispatcher.
Denegación de Servicio: Si esta organización se dedica a denegar los votos de todos o ciertos usuarios los usuarios pueden notificar esto ya que pueden demostrar que tienen TokenV y que no se ha guardado el voto hecho, esto es auditable por parte de la administración

Análisis de Ciberataques al Sistema En esta sección haremos un análisis de algunos de los ciberataques posibles por parte de una entidad malintencionada al sistema de votado, el objetivo de esta sección es buscar vulnerabilidades como si nosotros mismos fuésemos los atacantes para así encontrar posibles agujeros de seguridad.

Packet Replay: La táctica de packet replay consiste en, sin tener que ser capaces de descifrar el contenido de un mensaje, hacer copias de paquetes de una comunicación legítima para enviarlos posteriormente e intentar sacar algún beneficio de ello. ¿Por qué es este esquema seguro contra el Packet Replay? La parte de la implementación que nos garantiza esta seguridad es el protocolo TLS/SSL, como está explicado en el Anexo, TLS/SSL hace que, además de cifrar todos los paquetes y añadirles checksums apropiados, añade números de secuencia a los paquetes, estos números van cifrados también junto con todo el paquete por lo que no puede ser modificado por un atacante. Si un atacante, por ejemplo, quisiese interceptar mi conexión para obtener un TokenPS o un TokenV mediante esta táctica, tendría que, para empezar, interceptar mi conexión para copiar los paquetes, y bloquear mi conexión, pues si permite que mi conexión siga en pie, yo voy a recibir mi Token de forma correcta. Si un atacante hace esto, se queda con mi paquete legítimo y yo no recibiré nada, si ahora envía el atacante en la misma conexión este paquete, supongamos el paquete más crítico, el paquete que adjunta la prueba de pertenencia al grupo. Si el atacante lo envía en esta misma conexión, el votante verá su TokenPS activado, y el atacante no lo conocerá porque no le interesa hacer esto, si por el contrario se "guarda.[el] paquete con intención de utilizarlo más tarde para obtener él un TokenPS no puede, pues cuando inicie la conexión con el servidor, la clave simétrica que establece TLS/SSL va a ser diferente a con la que se ha cifrado el paquete con probabilidad prácticamente 1. Si por el contrario, al atacante le toca la lotería, este paquete es inútil, pues el protocolo establece una prueba dinámica de pertenencia al grupo, es decir, el atacante poseería una prueba de pertenencia al grupo con una pregunta que no se le ha hecho, i.e. un TokenPS distinto por lo que este ataque sería inútil.

SQL Injection: Ver Anexo K. Por lo descrito en el protocolo del sistema, cada uno de los servidores va a utilizar una base de datos local, como viene explicado en el Anexo X, siempre que se trate de contenido ajeno al propio servidor que vaya a ser ejecutado en una base de datos puede ser un agujero enorme de seguridad por temas de SQL Injection. Es por esto por

lo que todas las cadenas recibidas por parte del cliente se sanean siempre antes de ejecutarlas para evitar este tipo de ataques. Además el usuario que ejecuta cada uno de los servidores para acceder a sus propias tablas no tiene ningún tipo de permiso sobre el resto de la base de datos del sistema.

Man in the Middle Attack: Ver Anexo M. Este ataque, como viene explicado en el Anexo X, se basa en hacer de intermediario en una conexión legítima para ver y modificar el tráfico a gusto del atacante, gracias a que este sistema incorpora TLS/SSL y la implementación de certificados digitales, no es viable hacer un ataque de este estilo.

DoS/DDoS: Ver Anexo L. El ataque DoS se puede solucionar mediante un simple uso de un firewall con detección de ataques de Denegación de Servicio lo soluciona, ya que al ser ataque DoS todas las solicitudes vienen del mismo equipo y es fácil de detectar. Sin embargo con DDoS no es tan sencillo ya que no hay modo efectivo de impedir solicitudes, aún siendo erróneas, si el número de atacantes de la célula DDoS es suficientemente grande. Obviamente el impacto dentro del sistema está reducido al mínimo ya que en todas las comunicaciones con los servidores se comprueba la validez de una firma grupal y si no pasa, no se atiende la petición, sin embargo si pasa, significa que el ataque se realiza desde un subgrupo de votantes autorizados y una auditoría podría revelar su identidad.

3.1.3. Análisis de Anonimato del Voto

En este esquema se producen intercambios de información que, de ser mal realizados podrían comprometer el anonimato y la seguridad de un votante. Veamos que tipo de información se intercambia en cada momento de la comunicación y como así se puede asegurar que el voto es seguro y anónimo:

- Cuando el votante se comunica con el servidor G1 el servidor G1 conoce la identidad de este votante y sabe a que votante se le ha transmitido TokenPS, es decir, el servidor G1 puede realizar la correspondencia Votante⇔Pseudónimo
- Cuando el votante se comunica con el servidor TokenV Dispatcher, este solo conoce el TokenPS de este votante, el cual no revela ninguna información del votante.
- Cuando TokenV Dispatcher se comunica con G1 para verificar la validez del pseudónimo, G1 puede inferir a qué usuario se le va a emitir un TokenV, pero no puede averiguar qué TokenV es pues TokenV Dispatcher en ningún momento le otorga a G1 esta información. Asimismo TokenV Dispatcher sabe a que TokenPS le otorga qué TokenV y que la persona detrás de este TokenPS está debidamente autorizada para votar, pero no sabe quién es pues esta información reside únicamente en G1 y G1 no se la otorga.
- Cuando se publica el listado de votos G1 no puede averiguar de quién es cada voto pues la única identificación que acompaña al voto es TokenV.
- Del mismo modo A no puede averiguar la identidad pues lo máximo que puede asociar a un voto es el TokenPS del votante el cual no revela información del mismo.
- Como medida de seguridad adicional, el tráfico que vaya del cliente al servidor puede ir encriptado y enrutado mediante TOR, Ver AnexoN, o incluso mediante IPSEC como se explica en el artículo de Omar Lengerke y Raúl Barreño [34]. De este modo se garantiza que ni siquiera un análisis de tráfico de los votantes revela nada de la identidad del votante. ¿Por qué es esto importante? Si ponemos por ejemplo el caso de la Escuela de Informática de la Universidad, si se analiza el tráfico entrante del servidor de voto y vemos que hay muchos votos que vienen desde los laboratorios de la Escuela se puede inferir que con alta

probabilidad, el votante es estudiante y no profesor. Esto no revela la identidad del votante pero su grupo de anonimato ha sido reducido. Si nos vamos a un ejemplo más radical, es muy poco probable que un estudiante o profesor vote desde los equipos de secretaría, es por esto que un uso de una Red Anonimizadora como lo es TOR aporta funcionalidad deseable

3.1.4. Auditoría del voto

Una vez realizado el voto una de las características más importantes del voto electrónico es poder confirmar que el proceso se ha hecho de forma correcta y saber que el voto no ha sido modificado de ningún modo. ¿Cómo logramos esto en este esquema? Para empezar cuando un usuario ha votado correctamente se le envía un Hash del voto que lo identifica y además puede guardar toda la información generada en el proceso, de este modo un usuario puede verificar que su voto sigue intacto. Es decir, el usuario puede verificar criptográficamente que su voto final sigue intacto, pero gracias a las firmas grupales, tambien puede verificar que el proceso de votado es correcto, veamos como:

- El usuario puede estar seguro de que su token pseudónimo no lo tiene nadie más, al final el token pseudónimos sirve únicamente para pedir un token de votado, este token pseudónimo supongamos que es una cadena aleatoria alfanumérica de 128 caracteres. la probabilidad de que alguien pueda generar este token de forma aleatoria es aproximadamente de 1^{-200} por lo que podemos afirmar que en probabilidad este pseudónimo no lo puede tener nadie mas.
- Ahora bien, como hemos dicho, el proceso de votado electrónico debe ser auditable, tanto por el propio usuario como por la organización, ¿Qué garantía tiene el votante de que va a poder reclamar ese TokenPS como suyo si la situación lo demandase? Porque hemos visto que un usuario puede reclamar con una cadena identificativa de su voto que efectivamente un voto le pertenece. ¿Pero puede demostrar también que un Token Pseudónimo le pertenece? Si así fuese, un usuario puede auditar el 100 % del proceso de votado ya que puede reproducir los pasos intermedios. Gracias a las firmas grupales un usuario puede demostrar que un TokenPS es suyo, dado que la firma grupal que genera del mismo es única, y puede comunicarse con G1 para verificar que su TokenPS es válido
- Un usuario también puede demostrar que un TokenV es suyo y de nadie más ¿ Cómo hace eso? En las firmas grupales se puede emitir una prueba de que es el dueño de una firma digital, de este modo puede demostrar criptográficamente que una entrada [TokenV, groupsig(TokenPS)] le pertenece emitiendo una prueba de autoría de la firma digital groupsig(TokenPS). De este modo un usuario demuestra a la vez que es el dueño de TokenPS pues solo el dueño de este token puede generar groupsig(TokenPS) y demuestra ser dueño de la entrada [TokenV, groupsig(TokenPS)] por tanto es el dueño de TokenV tambien. Con esta demostración también puede demostrar que un voto le pertenece, pues es capaz de generar la ZKP de que es el dueño de un tokenV y voto pues el servidor guarda la firma grupal de este voto.

Sin embargo otro aspecto muy importante del voto electrónico es que bajo demanda es 100 % auditable por parte de la administración, es decir se puede romper el anonimato y privacidad si existe un acuerdo entre los organismos del voto. ¿Cómo funcionaría esto? Si El servidor G1 y A se ponen de acuerdo el servidor G1 puede darle la identidad de todas las firmas grupales que A almacena y de este modo ambos pueden asociar qué persona ha votado qué.

Esta sería una auditoría sin la colaboración del votante puesto que en ningún momento se requiere. Esto es deseable ya que los sistemas de votado electrónico son muy susceptibles de ser

atacados y bajo circunstancias excepcionales este anonimato se puede romper para comprobar el proceso del mismo. Sistemas como Helios Voting o NVoting, que son servicios de votado para particulares y empresas, siempre priorizan la integridad sobre la privacidad, y una de las primeras condiciones de Helios Voting, por ejemplo, es establecer un administrador, que bajo demanda va a poder abrir la votación y revelar la identidad de los votantes y sus votos. En el caso de esta auditoría general nosotros partimos con que la administración es la que posee esta autoridad, pero esta autoridad está dividida, es decir, si un organismo de el esquema quiere romper el voto, no va a poder.

Auditoría Individual del Voto En esta sección veremos como un usuario puede verificar y demostrar ser el dueño de un TokenPS, de un TokenV y de un Voto. Esto se consigue mediante las pruebas incluidas en la librería libgroupsig para emitir pruebas de conocimiento cero sobre la autoría de una firma grupal. El servidor A guarda, entre otras cosas la firma grupal del tokenPS emitida por el usuario al que se le ha dado este token pseudónimo. Si un usuario declara que él es el dueño de este token pseudónimo un protocolo para comprobarlo sería el que viene reflejado en la figura 3.2:

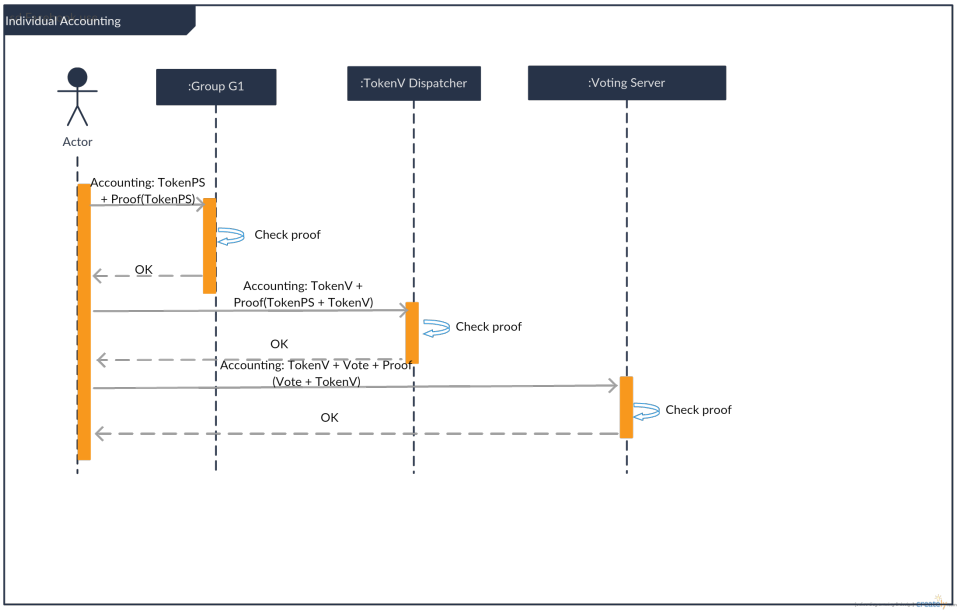

Figura 3.2: Auditoría Individual del Voto

Descripción del funcionamiento del protocolo de auditoría de voto electrónico.

1. El usuario quiere hacer una auditoría de su TokenPS, para auditar ser el dueño del mismo. El votante posee la clave de miembro de grupo que se ha usado a la hora de obtener el TokenPS, por lo que puede emitir la firma digital de este TokenPS, del mismo modo, con esta información puede emitir la ZKP de la autoría de la firma. Por lo que envía a G1 el TokenPS de la auditoría junto con la ZKP de la autoría de la firma.

2. El servidor G1 comprueba en su base de datos si existe ese TokenPS, de ser así recupera la firma digital que tiene guardada y contrasta la prueba de autoría enviada de la firma digital almacenada y responde con el resultado de la auditoría.

3. El votante va con su TokenPS que ha demostrado que es suyo, ahora queda demostrar que es el dueño de su TokenV. Enviando el TokenV y la prueba de autoría de la firma grupal

que guarda también TokenV Dispatcher que en este caso es de TokenPS y TokenV obtiene una auditoría de ser dueño de ese TokenV.

4. Del mismo modo que en los apartados anteriores, el servidor de voto guarda una firma grupal de voto + TokenV, utilizando la Prueba de Conocimiento Cero de autoría de firma grupal de un TokenV + Voto un usuario puede demostrar y comprobar que sus datos de voto han sido almacenados de forma correcta. Le envía esta prueba generada al servidor junto con el voto realizado y su TokenV, entonces el Server V comprueba la veracidad de esta prueba y responde al votante con el resultado de la auditoría.

Gracias a este método de auditoría solo el usuario dueño de la clave de miembro apropiada podrá comprobar que su TokenPS, su TokenV y su voto están almacenados correctamente y que el proceso se ha realizado de forma correcta, usando para demostrar su autoría las ZKP. Sin embargo al emitir el voto su identidad es anónima por lo que su auditoría deberá hacerse con lo que le haya propiciado el servidor de voto en el momento de votar de forma anónima. Esto se consigue de forma sencilla gracias a la posibilidad de emitir ZKP sobre firmas grupales ya que así el votante no revela ningún tipo de información a la hora de votar.

Auditoría Global del Voto En el voto electrónico, la integridad y auditoría es más relevante que el más absoluto anonimato, así lo reflejan sistemas como N-voting o Agora Voting en los cuales siempre existe un administrador de voto el cual puede abrir el voto de forma global. Gracias a como se ha diseñado este protocolo eso es posible, pero a diferencia con los sistemas de voto mencionados con anterioridad, aquí no existe un único administrador con esa capacidad, este protocolo necesita del acuerdo entre los miembros de la organización para poder abrir la votación.

Si hubiese que hacer una auditoría global del voto, i.e. Revelar todos los votos y las identidades de los votantes, lo único que habría que hacer es, gracias al diseño empleado, un simple join de las bases de datos de Server G1 y de TokenV Dispatcher ya que en ambos esta el campo TokenPS. Así Server G1 asocia la identidad de un votante con su TokenPS y TokenV Dispatcher la identidad de un TokenPS con su TokenV correspondiente, luego solo habría que mirar la lista de votos y asociar el TokenV con el voto. Este proceso es muy sencillo, pero es importante notar que este proceso es fácil solo gracias al diseño y que depende del acuerdo de dos autoridades diferentes, si no no es posible realizarlo. Esto es una mejora respecto a otros sistemas convencionales de voto ya que en ellos generalmente se delega toda la autoridad de realizar una auditoría en una sola entidad, de este modo la decisión depende de solo una persona y si se abre el voto podría ser que nadie se diese cuenta y el administrador ha roto el anonimato de la gente. Aquí eso es imposible gracias a que una entidad sin la colaboración de la otra es incapaz de abrir los votos.

3.2. Requisitos y Casos de Uso

En esta sección se recogen los requisitos funcionales y no funcionales del sistema que consideramos más importantes así como los casos de uso del mismo.

3.2.1. Requisitos Funcionales

- **RF1: Autorización de Voto.** Solo los usuarios autorizados a votar podrán completar el proceso de votado de forma correcta.

- **RF2: Auditoría de Voto Individual.** Los usuarios que hayan votado podrán realizar una auditoría individual de todo el proceso de su voto sin romper su anonimato.

- **RF3: Auditoría Gloval de Voto.** La administración puede realizar una auditoría global del voto pero no pueden hacerlo sin ponerse de acuerdo.

- **RF4: Orden de Paquetes de Conexión.** Los mensajes de los clientes tienen que llegar en orden correcto.

- **RF5: Cifrado de la conexión.** Toda conexión tiene que ir debidamente cifrada para garantizar la seguridad del protocolo.

- **RF6: Detección Voto Múltiple.** El sistema debe detectar un intento de voto múltiple por parte de un votante legítimo.

- **RF7: Anonimato.** El sistema debe preservar el anonimato del votante durante todo el proceso de votado.

- **RF8: Recuperación ante Pérdidas de Conexión.** El sistema de voto será resistente a pérdidas de la conexión i.e. El programa debe permitir votar a un votante después de que se pierda su conexión con el sistema, sea cual sea el momento.

3.2.2. Requisitos No Funcionales

- **RNF1: Conexiones Múltiples.** El sistema debe de permitir múltiples conexiones con intención de voto o de auditoría.

- **RNF2: Eficiencia.** El sistema no debe presentar una sobrecarga significativa.

- **RNF3: Recuperación Ante Errores.** El sistema debe poder recuperarse ante fallos de ejecución o del sistema.

3.2.3. Casos de Uso

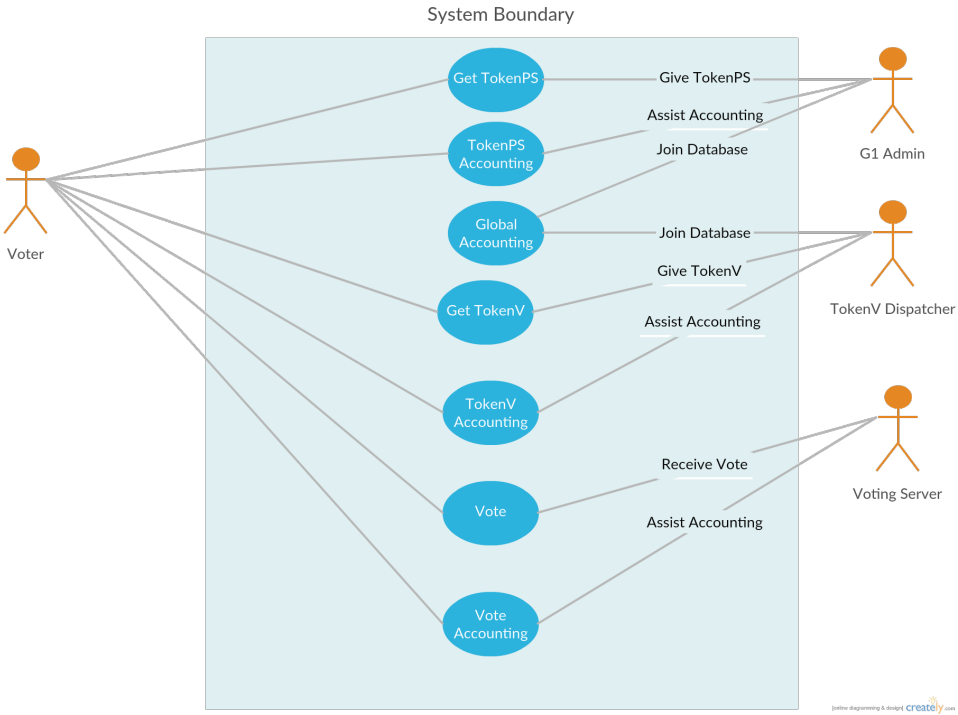

Figura 3.3: Diagrama de Casos de Uso

3.3. Codificación

La librería libgroupsig(*https://bitbucket.org/jdiazvico/libgroupsig/wiki/Home*) está implementada en C, esta librería cuenta con un toolset que encierra todas las funcionalidades de las firmas grupales, las cuales se han usado ampliamente para la prueba de concepto en este trabajo. Se encuentra descrita en el Anexo C. estos programas pueden ser invocados mediante una terminal de comandos UNIX. Dado que la parte más crítica del sistema, computacionalmente hablando, es la parte de justo las firmas grupales, y dado que existe este toolset, se ha decidido implementar los servidores y el cliente a utilizar en la prueba de concepto en Python, haciendo llamadas a este toolset cuando se quieran emplear firmas grupales. De este modo aprovechamos la eficiencia del toolset a la hora de utilizar las funciones de firmas grupales, y la flexibilidad que tiene Python para realizar las fases del protocolo.

3.3.1. Estructura del Sistema

En este trabajo se han desarrollado 4 módulos de un sistema de voto electrónico, estos sistemas se ejecutarán de modo independiente en diferentes servidores, pudiendo estar estos en la misma subred o en redes diferentes. Cada sistema contará con una base de datos local que utilizará para la correcta ejecución del protocolo. Los servidores a utilizar son:

- Client: Sistema que se encargará de que cada cliente pueda votar. Este programa debe existir en cada uno de los equipos que se utilicen para votar. Su código se encuentra en O.1.

- Server_G1: Sistema que se encargará de autenticar a los usuarios del sistema y les entregará su TokenPS. Cumple las funciones de Server G1. Su código se encuentra en O.2.
- Server_A: Sistema que se encargará de dar un Token de votado TokenV a cada TokenPS. Cumple las funciones de TokenV Dispatcher. Su código se encuentra en O.3.
- Server_V: Sistema que almacenará los votos de los votantes. Cumple las funciones de Server_V. Su código se encuentra en O.4.

El código de estos programas se encuentra en el Anexo O.

3.3.2. Bases de Datos

Como se ha mencionado con anterioridad el sistema desarrollado cuenta con 3 bases de datos independientes, cada base de datos será de tipo local en cada uno de los módulos del sistema global. Las bases de datos se han programado en MySQL y se acceden mediante los programas mediante la libreria de MySQL para Python. Cada base de datos será accedida solo por el servidor que la posea. Las tablas utilizadas en cada una de las bases de datos son las siguientes:

G1 Tabla Tokens:

Campo	Tipo
TokenPS(PK)	Varchar(128)
GroupSign	Varchar(4096)
Id	int(11)
Timestamp	timestamp

Cuadro 3.1: Tabla de la Base de Datos de G1

TokenV Dispatcher Tabla Tokens:

Campo	Tipo
TokenV(PK)	Varchar(128)
TokenPS(PK)	Varchar(128)
GroupSign	Varchar(4096)
Timestamp	timestamp

Cuadro 3.2: Tabla de la Base de Datos de TokenV Dispatcher

Voting Server Tabla Votes:

Campo	Tipo
Vote	Varchar(128)
TokenV(PK)	Varchar(128)
GroupSign	Varchar(4096)
Valid	bit(1)
Timestamp	timestamp

Cuadro 3.3: Tabla de la Base de Datos de Voting Server

Puesto que todos los servidores que guardan datos en este protocolo, que vienen a ser todos los servidores del sistema, los guardan en una base de datos. Es importante tener en cuenta

que los accesos a la base de datos los hace únicamente el servidor dueño de esta base de datos, como medida de seguridad adicional, para este esquema se ha hecho que la base de datos de cada servidor esté en el mismo, esto podría no siempre ser así, pero el hecho de que no exista ningún tipo de conexión externa a una base de datos aporta seguridad. Esta base de datos para esta implementación solo admite conexiones locales, es decir, que alguien, aún si hubiese podido hacerse con la contraseña de la base de datos, no podría acceder sin tener una sesión en la máquina en cuestión.

Estas bases de datos utilizan siempre datos que vienen del cliente por lo tanto es fundamental tener cuidado al ejecutar las consultas a las bases de datos con los datos otorgados por el cliente. Esto lo conseguimos gracias a que escapamos todos los caracteres no alfanuméricos para que sean tratados como una cadena y no como una sentencia SQL para evitar un ataque por SQL Injection.

4

Experimentos Realizados y Resultados

En esta sección se describen los experimentos realizados al esquema de votado electrónico explicado en la sección 3. Por la naturaleza de este proyecto y un marco de tiempo no muy extenso para la realización de pruebas la mayoría de experimentos corresponden a experimentos de pruebas de funcionamiento correcto y no pruebas de resultados numéricos como tiempos de respuesta o de ejecución.

4.1. Experimentos del sistema completo

En esta sección se explicarán los experimentos realizados sobre el sistema completo.

4.1.1. Pruebas de Funcionamiento Correcto

Como se ha comentado antes la mayoría de experimentos realizados son para comprobar el correcto funcionamiento del protocolo y la seguridad del mismo así como del anonimato. Esta sección cubre algunos de los experimentos realizados que consideramos más importantes.

- **Descripción:** Votado correcto, un voto realizado por un votante legítimo que no ha emitido su voto.
  Resultado: El sistema registra el voto de forma correcta y el cliente es notificado del mismo.
- **Descripción:** Auditoría correcta, un votante realiza una auditoría al sistema de su voto una vez ha votado y posee la clave correcta.
  Resultado: Cada servidor notifica que la auditoría es satisfactoria.
- **Descripción:** Intento de múltiples votos, un votante intenta votar una vez ha emitido su voto de forma satisfactoria.
  Resultado: El servidor G1 le notifica que está revocado y que por tanto no puede emitir este voto.
- **Descripción:** Intento de múltple voto en paralelo por parte de un votante correcto. Un votante inicia el proceso de votado de forma paralela con el objetivo de gracias a un

problema posible de mala gestión del control de concurrencia insertar múltiples votos.
Resultado: El sistema gestiona la concurrencia de forma correcta, y uno de los hilos de voto consigue emitir el voto de modo satisfactorio mientras que en el resto de hilos el cliente es notificado que ya ha votado.

- **Descripción:** Intento de voto por parte de un cliente al que se le ha cortado la conexión en medio del proceso de voto.
  Resultado: En todos los escenarios posibles, i.e. Antes y después de recibir TokenPS, antes y después de recibir TokenV, etc... El votante el programa es capaz de detectar en que proceso de votado concluyó y lo retoma resultando en un voto satisfactorio.

- **Descripción:** Intento de inyección SQL por parte de un cliente malintencionado.
  Resultado: El sistema escapa todos los caracteres no alfanuméricos impidiendo de este modo cualquier ataque posible de inyección SQL.

- **Descripción:** Intento de secuestro de voto mediante replay de paquetes.
  Resultado: El sistema deshecha los paquetes maliciosos gracias a que TLS/SSL deshecha los paquetes repetidos gracias al número de secuencia.

- **Descripción:** Intento de auditoría de un votante que no ha completado su voto. Un votante pierde la conexión en determinado momento del voto, y quiere hacer una auditoría para comprobar que el momento en el que se le ha parado ha sido el correcto.
  Resultado: El sistema comunica de forma correcta cuándo ha fallado el voto, i.e. antes o después de obtener el TokenPS, TokenV, o realizar el voto.

- **Descripción:** Intento de voto/auditoría después del reinicio de alguno de los servidores. Durante el proceso de votado es posible que algún equipo necesite ser reiniciado, y este experimento sirve para comprobar que no se requiere de mayor configuración que arrancar el programa con los parámetros correctos.
  Resultado: Los procesos de voto y auditoría se realizan de modo correcto después del reinicio de los servidores.

- **Descripción:** Intento de voto masivo por parte de múltiples clientes. El objetivo de esto es comprobar que ocurre cuando las solicitudes de voto superan con creces la capacidad de la cola del sistema.
  Resultado: El sistema no da servicio a las conexiones entrantes una vez la cola de peticiones esté llena. Sin embargo una vez se vacíe los clientes pueden votar del modo habitual.

4.1.2. Simulación de Voto

En el siguiente experimento el objetivo es determinar cual es el tiempo medio de voto de un voto en situación de que el servidor esté mayormente desocupado, como es esperable en un sistema de voto electrónico de las características explicadas en este trabajo.

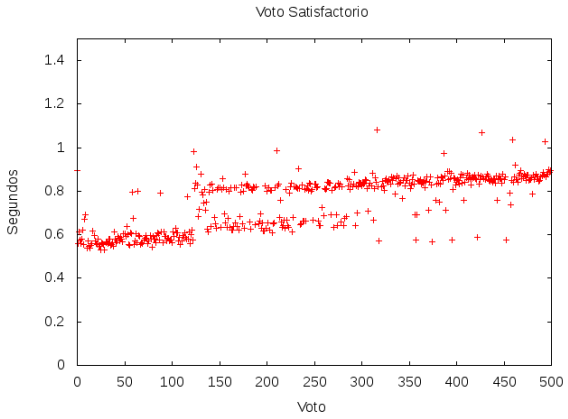

Figura 4.1: Simulación de Voto Satisfactorio

En la imagen 4.1 se recogen medidas de tiempo de 500 votos distribuidos idénticamente en el tiempo. Como vemos el tiempo de esta simulación realizada anda en torno a 0.7 segundos. Se observan 2 líneas de tendencia, la primera con un menor tiempo y la segunda con uno mayor, una explicación de esto puede ser la sobrecarga del sistema.

4.1.3. Voto Múltiple

En el siguiente experimento el objetivo es averiguar cuanto tiempo tarda el sistema en notificar a un votante que su voto ya ha sido emitido de forma satisfactoria.

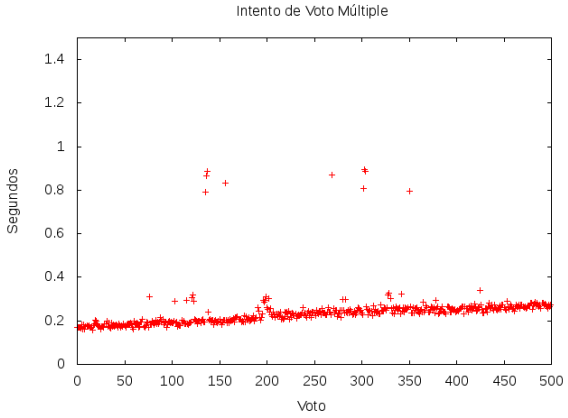

Figura 4.2: Tiempo de Voto Erróneo por Multiplicidad

En la imagen 4.2 se recogen medidas de tiempo de voto de personas que han intentado votar múltiples veces para medir cuanto tarda el sistema en notificar al usuario que él ya ha votado, este tiempo es del tipo 0.22s.

4.1.4. Auditoría Individual

En el siguiente experimento el objetivo es averiguar cuanto tiempo tarda el sistema en realizar una auditoría individual a un cliente de forma correcta.

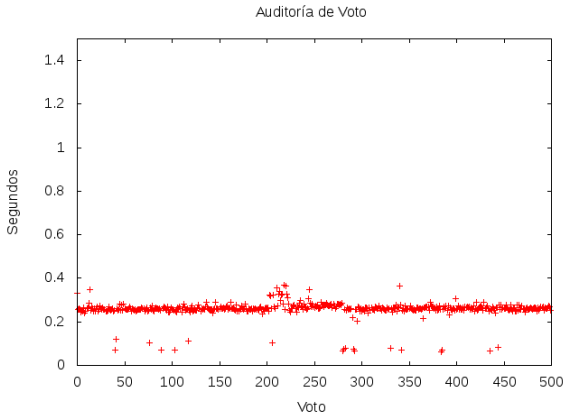

Figura 4.3: Tiempo de Auditoría Individual

En la imagen 4.3 se recogen medidas de tiempo de realización de auditoría individual correcta que anda en torno a los 0.25s.

4.1.5. Conclusiones

Como se ha mencionado estos experimentos están enfocados a realizar una comprobación del correcto funcionamiento del protocolo. En el futuro, como se recalca en 5.2 habría que realizar una simulación de voto en servidores distribuidos con múltiples votantes de diversas fuentes para poder realizar unas pruebas más realistas.

5

Conclusiones y trabajo futuro

5.1. Conclusiones

En este trabajo se ha hecho un estudio sobre tecnologías de cifrado moderno como son las firmas ciegas y las firmas grupales, y cómo podrían aplicarse a solucionar el problema del voto electrónico. Después de realizar un estudio de las mismas se diseñó un protocolo de voto electrónico basado en firmas ciegas (Ver Anexo J) y un protocolo basado en firmas grupales (Ver sección 3.1). Después de hacer un breve análisis de ambos, tanto en términos de seguridad como en términos de anonimato, la idea de poder realizar una auditoría del voto hizo fue el motivo principal por el que este trabajo se centrase más en las firmas grupales que las firmas ciegas. Además el protocolo de firmas ciegas me sirvió para entrenarme en el desarrollo de protocolos y en el estudio de estas tecnologías. Es por esto por lo que al final el análisis del protocolo basado en firmas grupales es más extenso y completo que el de firmas ciegas. Incluyendo una prueba de concepto a modo de implementación del protocolo.

El protocolo de votado elegido se ha implementado como prueba de concepto utilizando la librería libgroupsig([1]) haciendo llamadas a las funciones de su toolset desde Python. Este desarrollo se ha hecho en un entorno Linux(Debian Wheezy) desarrollando 4 programas independientes, 1 cliente para votar y 3 servidores acordes a lo explicado en 3.1. Después de su implementación se ha procedido a realizar diversos votos en diferentes pruebas, como se puede ver en el capítulo 4, véase realización de voto correcto, realización de voto con algún parámetro erróneo, intento de voto múltiple intentando explotar una posible mala gestión del control de la concurrencia. Y se ha comprobado que la prueba de concepto responde satisfactoriamente al estudio teórico.

Este sistema aprovecha completamente la funcionalidad que aportan las firmas grupales ya que se aprovecha el hecho de comprobar que una firma es correcta sin revelar la identidad como la revocación y la emisión de pruebas de conocimiento cero sobre firmas existentes.

El Protocolo desarrollado cumple a priori con todos los requisitos del votado electrónico además de proporcionar la funcionalidad de realizar una auditoría individual de todo el proceso sin revelar su identidad. Esto se ha probado mediante la implementación de la prueba de concepto con resultados satisfactorios. Sin embargo no se ha podido hacer un análisis más en detalle del protocolo para buscar posibles fallos de seguridad o vulnerabilidades, así como mediante el

estudio realizado este protocolo se puede considerar válido, no esta exento de poseer fallos de seguridad que se puedan encontrar a posteriori.

Este trabajo me ha servido para comprender lo complicado que es realizar un sistema de votado electrónico, los pros y los contras que conlleva y por qué le cuesta tanto a la sociedad llegar a implementarlo de forma generalizada. Se especifican algunos de los conocimientos más importantes que he obtenido durante este trabajo vienen explicados en el Anexo A.

5.2. Trabajo Futuro

Como trabajo futuro para este proyecto quedaría el establecer una autoridad certificadora en la que confíen todos los clientes y utilizar un cliente que acepte conexiones con servidores que tengan solo certificados de la autoridad de votado. Además queda hacer un análisis más en profundidad de la seguridad del protocolo utilizando herramientas más avanzadas y realizar un análisis más en profundidad de la anonimia, como viene explicado en [26]. Este análisis se podría hacer mediante el uso de una herramienta como π-calculus que permite un análisis de protocolos do modo formal.

Como siguiente tarea se debería probar este protocolo en un entorno más realista y más grande que las pruebas que se han realizado con el objetivo de llegar a implementarlo en un entorno universitario o empresarial. Una posibilidad sería utilizar servidores en una red universitaria y realizar una prueba con voluntarios para poder observar los resultados obtenidos y, si todo saliese bien, poder llegar a implementarlo en la propia universidad.

Además de lo mencionado anteriormente a este protocolo habría que realizarle un análisis de ciberseguridad más en profundidad que el realizado en 3.1.2. Comenzando por un análisis de las versiones de todo el software utilizado, incluyendo versión de Python, de MySQL, Linux, para poder encontrar vulnerabilidades específicas de esas versiones. Y acto seguido una auditoría de ciberseguridad en el que un auditor intenta atacar al sistema en un entorno de votación simulado.

Como trabajo futuro también se podría implementar una interfaz gráfica a cada uno de los programas para que sean más usables, en especial para los votantes ya que el programa actual cuenta con una interfaz de línea de comandos.

También se podría realizar una implementación del protocolo de modo que fuese independiente de la la plataforma, véase por ejemplo desarrollarlo mediante una plataforma web compatible con los navegadores más utilizados.

Glosario de acrónimos

- **RSA**: Algoritmo de cifrado de clave pública creado por Ron Rivest, Adi Shamir y Leonard Adleman
- **ECC**: Criptografía de Curvas Elípticas, es un criposistema de clave pública basado en el comportamiento de ciertas curvas elípticas sobre grupos finitos
- **Hash**: Función resumen de longitud de output reducida y conocida que sirve como función resumen de ficheros, cadenas, etcétera
- **Handshake**: Proceso mediante el cual un cliente y un servidor se ponen de acuerdo para establecer un canal de comunicación seguro
- **TLS**: Transport Layer Security, Seguridad de la Capa de Transporte, protocolo que opera a nivel de transporte estableciendo un cifrado seguro para la comunicación
- **SSL**: Secure Socket Layer, el predecesor de TLS que opera con seguridad a nivel de socket
- **ZKP**: Zero Knowledge Proof, Prueba de Conocimiento Zero, protocolo mediante el cual un miembro de la comunicación puede demostrar a un verificador con cierta confianza que conoce la solución a un problema complejo sin aportar ningún tipo de información sobre la solución del mismo
- **Token**: Ticket, en este ámbito Token se conoce como identificador reconocido por su emisor que se le otorga a un usuario con objetivo de utilizarlo y ser reconocido por esta organización como válido
- **TokenPS**: Token de Pseudónimo, cadena identificativa emitida por una autoridad con objetivo de ser usado como pseudónimo ocultando la identidad real del que lo posee
- **TokenV**: Token de Voto, cadena identificativa que acredita que el que la posea está autorizado a votar
- **DoS**: Denial of Service, ataque informático que consiste en generar mucho tráfico contra un servicio con intención de saturar los recursos del servicio con las peticiones del atacante haciendo que las peticiones de clientes legítimos no puedan ser atendidas
- **DDoS**: Distributed Denial of Service, ataque informático basado en realizar ataques DoS desde múltiples equipos
- **MitM**: Man in the Middle, Hombre en el Medio, ataque informático mediante el cual un atacante se sitúa como intermediario ilegítimo de una comunicación entre 2 equipos con intención de vulnerar de algún modo la seguridad de la comunicación, ya sea vulnerando la privacidad, o manipulando la comunicación
- **User Impersonation**: Impersonación de Usuario, acto de conseguir hacerse pasar por un usuario a efectos prácticos totalmente efectivos gracias a haberse hecho con información sensible del mismo(usuario/contraseña, clave privada)

- **OWF**: One Way Function, Función de un Solo Sentido, función que es computacionalmente viable calcular, pero para la cual calcular su inversa es computacionalmente inviable, por ejemplo la exponenciación en un grupo finito suficientemente grande
- **OTP**: One Time Pad, Libreta de un Solo Uso, algoritmo de cifrado en texto plano que hace uso de una sola contraseña para cifrar y descifrar y después esta no se vuelve a utilizar
- **VPN**: Virtual Private Network, servicio criptográfico que otorga una salida a internet desde una localización geográfica diferente
- **ISP**: Internet Service Provider, proveedor de servicio de conexión a Internet
- **TOR**: The Onion Router, red distribuida de baja latencia superpuesta a Internet que mantiene mediante ofuscación y redireccionamiento el anonimato de las comunicaciones. Se le conoce principalmente por ser la ruta de acceso a la Deep Web o Red Profunda y la Dark Net o Red Oscura
- **Deep Web**: Red Profunda, subconjunto de Internet conformado por los sitios web que no son indexados por los motores de búsqueda convencionales como Google, Yahoo o DuckDuckGo
- **Dark Net**: Red Oscura, subconjunto de la Deep Web que permanece inaccesible a todos los equipos a no ser que se posea de algún tipo de información adicional, esto puede venir dado por un secreto, software específico o protocolo seguido. Esta red es popular dado que es utilizada por organizaciones secretas, crimen organizado y organismos militares.
- $\mathbb{Z}_n$: Formalmente es el siguiente grupo cociente $\mathbb{Z}/n\mathbb{Z}$ donde $n\mathbb{Z} = \{n * z : z \in \mathbb{Z}\}$

Bibliografía

[1] Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. libgroupsig: An extensible c library for group signatures. 2015. http://eprint.iacr.org/2015/1146.

[2] David Chaum. *Blind Signature System*, pages 153–153. Springer US, Boston, MA, 1984.

[3] David Chaum and Eugène Van Heyst. Group signatures. In *Proceedings of the 10th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'91, pages 257–265, Berlin, Heidelberg, 1991. Springer-Verlag.

[4] Jonathan Katz, Steven Myers, and Rafail Ostrovsky. *Cryptographic Counters and Applications to Electronic Voting*, pages 78–92. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[5] Ron Rivest. Mix-net voting systems. http://courses.csail.mit.edu/6.897/spring04/L18.pdf, 2015. Accessed: 2017-03-24.

[6] Thomas Gert Ossler. *Electronic Voting over the Internet âĂŞ an E-Government Speciality.* PhD thesis, University of Technology Graz, Austria, 2007.

[7] Victor Manuel Morales Rocha. Seguridad en los procesos de voto electrónico remoto: Registro, votación, consolidación de resultados y auditoria. *Publication Services of the UPC Barcelona*, 2009.

[8] Christophe De Cannière. *Triple DES*, pages 1318–1319. Springer US, Boston, MA, 2011.

[9] Joan Daemen and Vincent Rijmen. *Rijndael/AES*, pages 520–524. Springer US, Boston, MA, 2005.

[10] Washington-Post. Nsa slides explain the prism data-collection program. http://www.washingtonpost.com/wp-srv/special/politics/prism-collection-documents/. Accessed: 2017-02-22.

[11] Marina Blanton. *Hash Functions*, pages 1287–1288. Springer US, Boston, MA, 2009.

[12] Antoon Bosselaers. *Md4-Md5*, pages 378–379. Springer US, Boston, MA, 2005.

[13] *RSA Cryptosystem, Primality, and Factoring*, pages 311–359. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[14] Victor S. Miller. *Use of Elliptic Curves in Cryptography*, pages 417–426. Springer Berlin Heidelberg, Berlin, Heidelberg, 1986.

[15] Torben Pedersen. *PKIX, Public Key Infrastructure (X.509)*, pages 939–940. Springer US, Boston, MA, 2011.

[16] Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. New x.509-based mechanisms for fair anonymity management. *Science Direct*, pages 111–125, 2014.

[17] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.

[18] Feng Wang Huixin Wu. A survey of noninteractive zero knowledge proof system and its applications. *Hindawi http://dx.doi.org/10.1155/2014/560484*, 2014:7, 2014.

[19] Adam Bender, Jonathan Katz, and Ruggero Morselli. *Ring Signatures: Stronger Definitions, and Constructions Without Random Oracles*, pages 60–79. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[20] Aggelos Kiayias, Yiannis Tsiounis, and Moti Yung. *Traceable Signatures*, pages 571–589. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[21] Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. *Anonymity Revocation through Standard Infrastructures*, pages 112–127. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[22] Seung Geol Choi, Kunsoo Park, and Moti Yung. Short traceable signatures based on bilinear pairings. pages 88–103, 2006.

[23] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. pages 41–55, 2004.

[24] Tatsuaki Okamoto. *An electronic voting scheme*, pages 21–30. Springer US, Boston, MA, 1996.

[25] Andreu Riera. Comments on the report e-voting security study. Technical report, Scytl Online World Security, S.A., 2002.

[26] Triinu Magi. Practical security analysis of e-voting systems. 2009.

[27] Marco Ramilli. *Designing A New Electronic Voting System.* PhD thesis, University of Bologna, 2007.

[28] Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. On securing online registration protocols: Formal verification of a new proposal. *Know.-Based Syst.*, 59:149–158, March 2014.

[29] Panagiotis Grotas. Secure multi party computations for electronic voting. *University of Athenas Thesis Publications*, 2014.

[30] Notes on cryptography. `https://crypto.stanford.edu/pbc/notes/crypto/voting.html`, 2016. Accessed: 2017-03-20.

[31] L. Malina, J. Smrz, J. Hajny, and K. Vrba. Secure electronic voting based on group signatures. In *2015 38th International Conference on Telecommunications and Signal Processing (TSP)*, pages 6–10, July 2015.

[32] Lukas Malina, Jan Smrz, and Jan Hajny. Electronic elections based on group signatures. *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems*, 5(1):15–21, 2016.

[33] L. Malina, J. Smrz, J. Hajny, and K. Vrba. Secure electronic voting based on group signatures. In *2015 38th International Conference on Telecommunications and Signal Processing (TSP)*, pages 6–10, July 2015.

[34] Omar Lengerke Raul Bareno Gutierrez. Voto electrónico con ssl/tls e ipsec. 2005.

[35] Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. Fair anonymity for the tor network. *arXiv:1412.4707v1*, 2014.

[36] Kenneth J. Arrow, Leonid Hurwicz, and Hirofumi Uzawa. *An Improvised Protocol For Demonstrating Possession Of Discrete Logarithms And Some Generalizations*. PhD thesis, Centre for Mathematics and Computer Science, Kruislaan 413 1098 SJ Amsterdam The Netherlands, 1988.

[37] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, MichaÃńl Quisquater, Louis C. Guillou, Marie Annick Guillou, GaÃŕd Guillou, Anna Guillou, Gwenolé Guillou, Soazig Guillou, and Thomas A. Berson. How to explain zero-knowledge protocols to your children. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 628–631. Springer, 1989.

[38] David Arroyo, Jesus Diaz, and Francisco B. Rodriguez. *Non-conventional Digital Signatures and Their Implementations—A Review*, pages 425–435. Springer International Publishing, Cham, 2015.

[39] Jan Camenisch and Markus Stadler. *Efficient group signature schemes for large groups*, pages 410–424. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[40] Ronald L. Rivest, Adi Shamir, and Yael Tauman. *How to Leak a Secret*, pages 552–565. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[41] Martin Hirt. *Multi-Party Computation: Efficient Protocols, General Adversaries, and Voting*. PhD thesis, ETH ZURICH, 2001.

[42] Jorg Helbach, Jorg Schwenk, and Sven Schage. Code voting with linkable group signatures. In *Electronic Voting*, 2008.

A

Conocimientos Adquiridos

Durante este trabajo he adquirido, entre otros, los siguientes conocimientos:

- Conocimiento del estado de la criptografía a día de hoy.
- Conocimiento de las firmas grupales y sus aplicaciones.
- Conocimiento de los requisitos fundamentales para que un sistema de votado se considere anónimo y seguro.
- Conocimiento de protocolos actuales utilizados en votado electrónico.
- Funcionamiento de las firmas ciegas.
- Funcionamiento de las firmas de anillo.
- Funcionamiento de las redes de anonimato tipo TOR.
- Conocimiento sobre las pruebas de conocimiento cero y sus posibles aplicaciones así como su trasfondo matemático.
- Funcionamiento de la librería libgroupsig.
- Funcionamiento de TLS/SSL.
- Funcionamiento y metodología de los ataques informáticos más populares, SQL Injection, DDoS, Man in the Middle etc...
- Conocimiento sobre la búsqueda y lectura de documentación científica.
- Conocimiento sobre técnicas de análisis de la privacidad y de la seguridad de protocolos de comunicaciones.

B

Manual de utilización

En esta sección se explicará como utilizar los programas desarrollados para implementar el protocolo explicado en 3 cuyo código se encuentra en el Anexo O.

B.1. Voting Client

Para utilizar el programa de votado hay que ejecutar la siguiente línea de comandos:

python <ruta al programa> <IP del Servidor G1> <Puerto del servicio del Servidor G1> <IP del Servidor A> <Puerto del servicio del Servidor A> <IP del Servidor V> <Puerto del servicio del Servidor V> <Clave Privada de Miembro de Grupo> <Clave Pública de Grupo> {a | v} [voto a emitir]

Donde a o v significa auditoría o voto, y voto a emitir va con la opción de votado.

Acto seguido el programa procederá según lo solicitado, si los parámetros y ficheros son correctos el programa funcionará de forma correcta, si algún fichero o parámetro es incorrecto el programa fallará notificándolo así al usuario.

B.2. Server G1

Para utilizar el programa de votado hay que ejecutar la siguiente línea de comandos: **python <Ruta al programa> <IP del Servidor G1> <Puerto del servicio del Servidor G1> <Clave Privada RSA> <Certificado Digital> <Clave Pública de Grupo> <Clave de Administrador de Grupo> <CRL> <GML>**

Acto seguido el programa funciona de forma autónoma y correcta siempre que los parámetros se hayan introducido de forma correcta.

B.3. Server A

Para utilizar el programa de votado hay que ejecutar la siguiente línea de comandos:

python server_A.py <IP del Servidor A> <Puerto del servicio del Servidor A> <IP del Servidor G1> <Puerto del servicio del Servidor G1> <Clave Privada RSA> <Certificado Digital> <Clave Pública de Grupo>

Acto seguido el programa funciona de forma autónoma y correcta siempre que los parámetros se hayan introducido de forma correcta.

B.4. Server V

Para utilizar el programa de votado hay que ejecutar la siguiente línea de comandos:

python server_V.py <IP del Servidor V> <Puerto del servicio del Servidor V> <IP del Servidor A> <Puerto del servicio del Servidor A> <Clave Privada RSA> <Certificado Digital> <Clave Pública de Grupo>

Acto seguido el programa funciona de forma autónoma y correcta siempre que los parámetros se hayan introducido de forma correcta.

C

Librería libgroupsig

En esta sección comentaremos que funciones hay en este toolset, como se ejecuta, y como se emplean en este esquema de voto electrónico.

1. **group_create**: Programa utilizado para la creación de un grupo de cifrado. Uso:
   group_create <scheme> -d <GS base dir> -M <GS Manager subdir> -g <GS group public data subdir> -m <GS members subdir> [-p <prime size>]
   Esta función se utiliza antes de que la votación tenga lugar, denotando el grupo que se va a utilizar.

2. **join**: Programa utilizado una vez el grupo se ha creado para añadir miembros, i.e. Generar las claves de miembros de grupo correspondientes. Uso:
   join <scheme> <group key> <mgr key> <gml> <members dir> [<n members> = 1]
   Este programa se utilizará al comienzo de la votación, en la que se establecen los miembros autorizados a votar, en este caso se generarían tantas claves como votantes y se enviarían de forma segura a cada uno de los votantes, utilizando por ejemplo, su correo universitario o corporativo.

3. **sign**: Programa utilizado para la generación de firmas grupales. Uso:
   sign <scheme> <signature file> <msg file> <member key> <group key>
   Este programa se utilizará por parte del cliente a la hora de pedir su TokenPS y de pedir su TokenV para que el servidor g1 pueda verificar su identidad y que el servidor A pueda verificar que es un miembro válido de grupo.

4. **verify**: Programa utilizado para comprobar la autenticidad de una firma grupal de un fichero. Uso:
   verify <scheme> <signature file> <msg file> <group key>
   Este programa se utiliza para que el usuario demuestre que es un miembro válido del grupo de cifrado.

5. **revoke**: Programa utilizado para revocar a un usuario que ha emitido una firma digital de grupo. Uso:
   revoke <scheme> <sig> <grp_key> <mgr_key> <GML> <CRL>
   Este programa se utiliza para revocar la firma de un usuario indicando que a este usuario ya se le ha emitido un TokenPS

6. **trace**: Programa utilizado para verificar si una firma viene de un miembro del grupo que ha sido revocado. Uso:
   trace <scheme> <sig> <grp_key> <CRL> <mgr_key> <GML>
   Este programa se utiliza cuando un usuario pide un TokenPS para verificar si se le ha otorgado ya uno o no.

7. **claim**: Programa utilizado para emitir una prueba de conocimiento cero de autoría de una firma digital. Uso:
   claim <scheme> <member key> <group key> <proof> <sig1> ... <sign>
   Esta función se utiliza a la hora de realizar una auditoría individual sobre el proceso de votado.

8. **claim_verify**: Este programa se utiliza para demostrar mediante una ZKP que una serie de firmas grupales han sido emitidas por cierto miembro del grupo. Uso:
   claim_verify <scheme> <proof> <group key> <sig1> ... <sign>
   Este programa se utiliza para verificar la autoría de una firma digital a la hora de realizar una auditoría.

9. **same_signer**: Este programa se utiliza para averiguar si una serie de firmas provienen del mismo emisor. Uso:
   same_signer <scheme> <sig1> <sig2> <grp_key> <mgr_key> <GML> <CRL>
   Este programa no se utiliza en el esquema de voto electrónico.

Los parámetros utilizados por estos programas son los siguientes respectivamente:

- **scheme**: Esquema de cifrado de grupo a utilizar, siendo estos BBS04, KTY04 y CPY06.
- **GS base dir**: Directorio base en el que se va a guardar la información del grupo.
- **GS Manager subdir**: Subdirectorio del directorio base en el que se va a guardar la información del administrador del grupo.
- **GS group public data subdir**: Subdirectorio en el que se guarda la información relevante que va a ser pública del grupo, en este caso la clave de grupo, recordar que con esta clave se pueden verificar firmas que procedan del grupo pero no emitir ninguna.
- **GS members subdir**: Subdirectorio en el que se guardarán las claves de miembros de grupo de los usuarios que serán usadas para firmar mensajes como miembros del grupo.
- **primesize**: Tamaño del número primo a utilizar en la generación del grupo, este argumento es opcional y solo válido para algunos esquemas de grupo.
- **group key**: Clave pública del grupo de cifrado.
- **mgr key**: Clave del administrador del grupo, solo el administrador es capaz de generar claves.
- **gml**: Fichero que almacena información parcial de cada una de las claves generadas para más adelante poder identificar y revocar firmas de modo correcto.
- **n members**: Número de claves de miembros de grupo a generar.
- **signature file**: Fichero en el que se guarda una firma digital.
- **msg file**: Fichero a partir del cual generar una firma.

- **GML**: Fichero con información parcial de los usuarios que se utilizará para identificar al emisor de la firma
- **CRL**: Fichero que guarda una lista con usuarios revocados del sistema.
- **proof**: Fichero que contendrá una ZKP.
- **sigi**: I-ésima firma digital sobre la que generar/verificar la prueba de autoría de emisión.

D

TLS/SSL

El caso más general de las comunicaciones de Internet se producen entre equipos que no se han comunicado con anterioridad antes, y mucho menos comparten un secreto con el que establecer un cifrado simétrico rápido. La criptografía de clave pública es mucho más lenta y costosa computacionalmente que la criptografía simétrica, y no es viable que toda la información en internet fluya usando criptografía asimétrica. Por lo tanto lo deseable sería encontrar un método por el que dos miembros de una comunicación lleguen a un acuerdo en la clave simétrica que usar de forma secreta, y eso es lo que TLS/SSL ofrece. A grandes rasgos TLS/SSL utiliza la Criptografía de Clave Pública solo para ponerse de acuerdo en qué cifrado simétrico usar y posteriormente pasar a usar este cifrado para el resto de la comunicación.

En detalle el algoritmo TLS/SSL funciona de la siguiente forma:

- El cliente envía el mensaje *Client Hello* en el que establece la comunicación con el servidor comunicándole que tipos de protocolos de clave pública y simétrica conoce
- El servidor contesta con el *Server Hello* en el que especifica que algoritmos de cifrado van a utilizar en el resto de la comunicación además de enviar su certificado digital
- El cliente envía su certificado digital
- Mediante la Criptografía de Clave Pública el cliente y el servidor negocian una clave simétrica a utilizar en el resto de la comunicación
- La comunicación de clave asimétrica se termina y se utiliza en adelante la criptografía de clave simétrica mediante la clave acordada

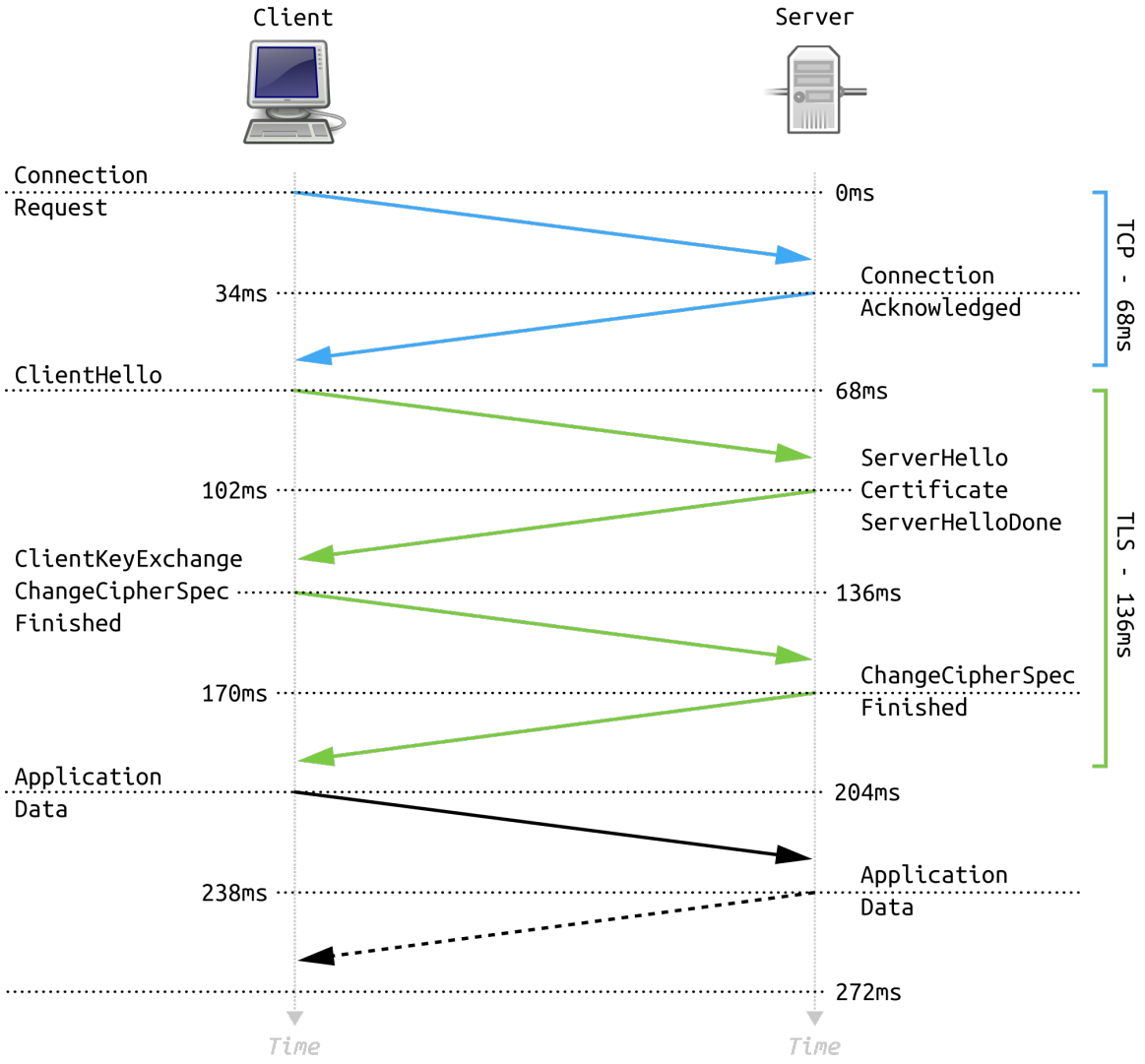

Figura D.1: TLS Handshake (Fuente: *www.wikipedia.org*)

Un ataque realizable a este protocolo es interceptar el *Client Hello* y suprimir todos los protocolos salvo el más fácil de atacar con motivo de que el servidor se vea forzado a elegir ese protocolo y poder intentar esnifar el tráfico cifrado que utilizarán en adelante.

Este esquema protege a ataques de replay ya que dentro de los mensajes hay un código de secuencia que va cifrado por lo que los mensajes que se pudiesen repetir se eliminan una vez se haya recibido el primero. El único ataque viable a SSL es, suplantando la identidad del cliente seleccionar un criptosistema más o menos vulnerable con objetivo de intentar posteriormente romper todos los paquetes capturados buscando la clave simétrica a utilizar. Es por esto que TLS/SSL se considera muy seguro, además de que este protocolo cuenta con versiones del mismo cada cual mas segura que el anterior.

TLS/SSL se utiliza en todos los servidores del proceso de votado ya que cuando se establecen comunicaciones por parte del cliente con el servidor de forma anónima se necesita seguridad y anonimato.

TLS/SSL forma parte fundamental del protocolo electrónico basado en firmas grupales. Ya que soluciona problemas de seguridad y anonimato a la hora de comunicarse de forma anónima. TLS/SSL no solo garantiza un cifrado para evitar que cualquiera en la red pueda ver el voto o el mensaje enviado. Esto protege de muchos ataques como de replay o secuestro de sesión.

E

Factorización de enteros en factores primos

Uno de los cÃąlculos mas repetidos e importantes de las matemÃątica ha sido la descomposición de un número en sus factores primos, los números primos siempre han sido protagonistas en las matemÃątica y han dado pie a muchos avances tecnológicos en la vida humana, uno de estos grandes avances es la criptografía. Veamos el problema formalizado matemÃąticamente.

P es primo $\Leftrightarrow \forall n \in \mathbb{N}(n|P \Rightarrow (n = P \vee n = 1))$

$p_1, p_2, ..., p_k$ es la descomposición en factores primos de $P \Leftrightarrow P = \prod_{i=i}^{k} p_i \wedge \forall i \in \{1, 2, ..., k\} p_i$ primo. Es un problema que no es complicado de resolver, pues se puede comprobar fácilmente si un número es primo, se puede hacer una lista de números primos menores que un determinado N y se puede comprobar si ese número divide a N. El problema dicho de este modo no suena complicado pero el tiempo de ejecución de este algoritmo es exponencial, por lo que si nos vamos a números grandes como números de 1024 bits en representación binaria, resulta inviable computacionalmente encontrar esta descomposición en factores primos.

Gracias a este hecho el algoritmo de clave pública más famoso, el RSA (Ver 2.2.4), es posible ya que si se encontrase un algoritmo capaz de encontrar la factorización en factores primos de un número arbitrariamente grande en tiempo polinómico, este sistema quedaría vulnerado en esencia.

F

Teoría de Grupos

La teoría de grupos es una rama de las matemáticas que se dedica a estudiar de forma abstracta grupos, sin importar qué es lo que se esté manejando para estudiar sus propiedades intrínsecas y propiedades estructurales que pudiese tener. La teoría de grupos es una teoría que se relaciona, y se podría decir que estÃą contenida en la teoría de cuerpos, la cual se relaciona muy estrechamente con la teoría de Galois. El objetivo de esta sección es explicar las nociones más fundamentales de los grupos, que grupos se suelen utilizar en criptografía y por qué.

Definición: Un grupo es un par $(G, \circ)$ que cumpla las siguientes propiedades:

1. **Asociatividad:** $\forall a, b, c \in G : (a \circ b) \circ c = a \circ (b \circ c)$
2. **Elemento neutro:** $\exists e \in G \forall a \in G : e \circ a = a \circ e = a$
3. **Elemento inverso:** $\forall a \in G \exists a^{-1} : a \circ a^{-1} = a^{-1} \circ a = e$

Unas consecuencias inmediatas de estas propiedades que se pueden deducir sin mucho esfuerzo es que el elemento neutro es único, así como el elemento inverso de cada elemento también es único. Vemos que no hemos especificado ni que conjunto ni que operación se realiza en estos conjuntos, veamos unos ejemplos sobre que tipos de grupos podemos encontrarnos:

1. El conjunto de lo números enteros módulo n siendo n entero, también representado como $\mathbb{Z}_n$ respecto a la operación suma.
2. El conjunto de matrices con coeficientes reales de determinante 1 respecto al producto de matrices usual.
3. El conjunto de biyecciones de un conjunto finito respecto a la composición de funciones.

Como vemos hay grupos de lo más variopintos y peculiares, en criptografía nos vamos a centrar en unos grupos muy particulares, y que son, como veremos ahora, en general muy buenos.

Definición: Un grupo G se dice abeliano $\Leftrightarrow \forall a, b \in G : a \circ b = b \circ a$ Esta propiedad es muy buena, ya que la conmutatividad simplifica muchas situaciones que se pueden dar a la hora de estudiar grupos. como hemos visto en los ejemplos anteriores, por ejemplo, ni el grupo 2 ni el

grupo 3 son abelianos ya que el producto de matrices no es conmutativo y tampoco lo es la composición de funciones.

Definición: El orden de un elemento a se define de la siguiente forma $ord(a) = n \Leftrightarrow a^n = e \wedge \forall n' : a^{n'} = e \Rightarrow n|n'$ Es decir, es el mínimo n tal que $a * a * ... * a = e$ siendo multiplicado n veces. Una propiedad del orden es la siguiente: Si $n = |G| \Rightarrow \forall a \in G\ ord(a)|n$. Una consecuencia inmediata es que si p=n es primo, todos los elementos tienen orden p salvo el elemento neutro

F.1. La Teoría de Grupos en la Criptografía

En esta sección veremos en que situaciones se utiliza la teoría de grupos en la criptografía moderna.

F.1.1. RSA

RSA es posible gracias a el Teorema de Euler. Antes de enunciar el Teorema de Euler veamos la función de Euler. La función ϕ es una función $\phi : \mathbb{N} \to \mathbb{N}$ que calcula el número de números naturales menores que el input n que son coprimos con n, esta función está definida de la siguiente forma.
$\phi(1) = 1$
$\phi(p^k) = (p-1)p^{k-1}$ si p primo
$\phi(m * n) = \phi(m) * \phi(n)$ si $mcd(m,n) = 1$
Sea $m \in N$: su descomposición en factores primos es $m = \prod_{i=1}^{j} p_i^{k_i} \Rightarrow \phi(m) = \prod_{i=1}^{j} \phi(p_i^{k_i})$

Teorema de Euler: Sean $a, n \in \mathbb{N} : mcd(a,n) = 1 \Rightarrow n|(a^{\phi(n)} - 1)$ O su equivalente utilizando teoría de grupos: Sea $a \in \mathbb{Z}_n : mcd(a,n) = 1 \Rightarrow a^{\phi(n)} = 1 \mod n$ ¿Por qué es importante el teorema de Euler en RSA? En RSA de manera muy inteligente se consigue lo siguiente. Cuando se aplican las dos claves e,d(elegidos de modo que e es inverso multiplicativo de d en aritmética modular $\phi(n)$) sobre un mensaje m en aritmética modular n queda lo siguiente, para algún k. $m^{e*d} = m^{k*\phi(n)+1} = (m^{\phi(n)})^k * m^1 = 1^k * m = m \mod n$

F.1.2. Logaritmo Discreto

El logaritmo discreto es una operación matemática que opera sobre elementos de un grupo cíclico finito $G = \mathbb{Z}_n$ del modo siguiente: $a = \log_x y \Leftrightarrow y = x^a \mod n$ Vamos a hacer unas comparaciones respecto al logaritmo discreto y la exponenciación en $\mathbb{R}$ y en $\mathbb{Z}_n$:
Exponienciación: La operación de la exponenciación resulta ligeramente más sencilla en aritmética modular, ya que, para empezar se opera con enteros y no hay potencias fraccionarias las cuales suponen cálculos más complejos, y que al estar utilizando la aritmética modular n. Las números a manejar nunca son más grandes que n^2
Logaritmo: En el análisis calcular un logaritmo no es tarea difícil, el logaritmo siempre es una función creciente, es más es una función creciente perteneciente a la clase C^∞ es decir, es infinitamente derivable con derivada contínua en su dominio de definición. Esto permite que calcular logaritmos sea sencillo en términos computacionales mediante muchos métodos númericos, ya sea el famoso Método de Euler, el Método de Runge Kutta, o el simple a la par que eficaz desarrollo en serie de Taylor. Sin embargo, en aritmética $\mathbb{Z}_n$ para empezar se pierde el análisis, aquí la noción de derivabilidad no existe, es más, una simple noción como que la función crece, o decrece no tiene sentido pues en aritmética modular justo el módulo pierde esta noción, aquí un valor muy grande si traspasa n se convierte en un valor muy pequeño. De hecho, el único

algoritmo que se conoce para el algoritmo discreto es ir multiplicando la base por si misma sistemáticamente hasta encontrar el resultado.

Es por esto por lo que el logaritmo discreto resulta de mucha utilidad en criptografía, expliquemos el por que brevemente, calcular una potencia modular en particular no es computacionalmente muy costoso, utilizando el algoritmo de potenciación binaria mediante cuadrados sucesivos es algo computacionalmente muy viable dada la reutilización de cálculos anteriores y los llamados cuadrados sucesivos. Sin embargo para calcular el logaritmo discreto de un número respecto a una base, el único algoritmo conocido es multiplicar la base por si misma contínuamente y comparando el resultado. Supongamos que nuestro grupo tiene orden primo de aproximadamente 1024 bits. Esto supondría que hay un total de aproximadamente 2^{1024} elementos, y encontrará por tanto el resultado a este algoritmo en un tiempo medio de $|G|/2 = 2^{1023} \approx 1{,}37 * 10^{362}$ Si suponemos que contamos con un Cluster capaz de procesar un cuadrillón de operaciones segundo 10^{20} tardaría $1{,}37 * 10^{342}$ Para hacer una comparativa de la magnitud de la que estamos hablando, la edad del universo es aproximadamente de $4{,}7 * 10^{40}$ segundos. Sin embargo con el algoritmo de potenciación binaria el número de operaciones a realizar es del orden $\approx \log_2(exp)$ donde exp es el exponente de la operación. Es decir, si A quiere calcular g^a siendo a un número de unos 1024 bits aproximadamente, para calcular la potencia solo tendrá que realizar 1024 operaciones aproximadamente. Y si alguien quisiese calcular el logaritmo discreto, como hemos explicado anteriormente, necesitaría del orden de 2^{1023}

F.1.3. Cifrado de ElGamal

El cifrado de ElGamal es seguro mientras el problema del logaritmo discreto sea computacionalmente irresoluble, para explicar esto vamos a explicar brevemente como funciona el cifrado de ElGamal.

Generación de Claves: A elige un número primo p tal que calcular un logaritmo discreto en $\mathbb{Z}_p$ sea computacionalmente imposible, por ejemplo como hemos visto antes, un primo de 1024 bits. Acto seguido escoge dos números aleatorios, g y a del grupo, g será el generador y a será la clave privada. A computa el valor de $K = g^a \mod p$ y utiliza como clave pública la tupla (g,p,K). Esto es seguro ya que hemos asumido que calcular a a partir de K y g es equivalente a calcular el logaritmo discreto que es computacionalmente inviable.

Cifrado: Supongamos que B le quiere enviar algo a A mediante este cifrado, B genera un número aleatorio b del grupo $\mathbb{Z}_p$ y si m es su mensaje a enviar, el mensaje cifrado a enviar sería $(g^b, K^b * m)$

Descifrado: con estos datos si A computa lo siguiente: $((g^b)^{-a}) * K^b * m = g^{-a*b} * g^{a*b} * m = 1 * m = m$ donde se ve que es necesario conocer a para poder descifrar este mensaje.

G

Suposiciones Criptográficas

En este anexo se mencionan y describen conceptos y suposiciones criptográficas en las que se basa la criptografía moderna, sin las cuales muchos de estos criptosistemas se verían completamente vulnerados.

G.1. Suposiciones Criptográficas sobre Funciones Hash

Estas propiedades se utilizan para definir la pseudo-inyectividad de las funciones hash. Por el principio de palomar es matemáticamente trivial que una funcion hash no puede ser inyectiva, pero se puede considerar pseudo-inyectiva si es computacionalmente inviable encontrar un ejemplo de no inyectividad, como se define más adelante.

En cuanto a las funciones Hash existen 2 tipos de condiciones relacionadas con la computación que son las siguientes:

1. **Resistencia a Colisión Débil:** Dada una función Hash H y un mensaje A es computacionalmente inviable encontrar B tal que $H(A) = H(B)$.

2. **Resistencia a Colisión Fuerte:** Dada una función Hash H es computacionalmente inviable encontrar A y B tal que $H(A) = H(B)$.

G.2. Decisional Diffie-Hellman

La suposición Diffie-Hellman Decisional está relacionada con la no correlación de los resultados de exponenciación dentro de un grupo cíclico (Ver Anexo F), esta suposición establece que siendo g un generador de un grupo G, i.e. $G =< g >$ es computacionalmente dificil establecer si una cadena recibida (A, B, C) es de la forma (g^a, g^b, g^{ab}) para $a, b \in \mathbb{N}$ ó (g^a, g^b, g^c) para $a, b, c \in \mathbb{N} : c \neq ab$

Ejemplo. Sea G = $\mathbb{Z}_7$0507 siendo 70507 primo. Por tanto cualquier elemento es generador. Sea 1024 un generador. y sean a=214, b=530 y c=274. La suposición diffie-hellman decisional nos dice que es computacionalmente difícil determinar si las tuplas siguientes: (55457, 11812,

57902) es tal que $\exists g, a, b : (g^a, g^b, g^{ab})$ Ã§ por el contrario $\exists g, a, b, c \neq a * b : (g^a, g^b, g^c)$. En este caso, es de la segunda forma.

G.3. Strong-RSA

La suposición RSA-Fuerte establece que siendo N un número compuesto de factorización desconocida y un número C < N no es viable computacionalmente encontrar un par (e,M) tal que $C = M^e$ mód N. Por ejemplo con N=705072, que es trivialmente compuesto. No es computacionalmente viable determinar a partir del nÃžmero 389056 un par (M,e) tal que $389056 = M^e$ mód 70572. Se puede comprobar que este par es (1024,35352).

G.4. q-Strong Diffie Hellman

El problema presentado en q-Strong Diffie Hellman consiste en, a partir de 2 grupos $G_1 G_2 : |G_1| = |G_2| = p \in \mathbb{P}$ y sean $g_1 g_2 :< g_1 >= G_1 < g_2 >= G_2$ generadores de los respectivos grupos y una entrada $(g_2, g_2^x, g_1, g_1^x, g_1^{x^2}, ..., g_1^{x^q}) \in G_2 x G_1^{q+1}$ obtener un par $(c, g_1^{\frac{1}{x+c}}) : x + c \neq 0 \ mod \ p$ es computacionalmente difícil.

H

Ejemplo de Prueba de Conocimiento Cero de la Cueva

Veamos el famoso ejemplo ilustrativo usado por Jean-Jacques Quisquater en su artículo Çómo explicar pruebas de conocimiento cero a tus hijos"[37] el cual me ha servido para comprender el concepto de prueba de conocimiento cero. Antes de nada establezcamos las premisas. Existe una cueva muy profunda que nada más entrar posee dos caminos que se acaban uniendo en forma de anillo, al final de los caminos existe una puerta que solo se abre bajo el conocimiento de unas palabras mágicas, Prover quiere demostrar que posee las palabras mágicas pero no quiere decírselas a nadie, entonces llama a Verifier para demostrárselo, pero Prover es vergonzoso y quiere que solo Verifier pueda verificar que él posee el secreto, Probador y Verifier proceden de la siguiente forma:

1. Probador se mete en la cueva y elige un camino al azar y lo recorre hasta llegar a la puerta mientras Verifier permanece fuera y no puede ver que camino ha elegido.

2. Probador grita desde el túnel a Verifier que entre, y Verifier entra justo hasta donde está la bifurcación del camino.

3. Verifier entonces elige un camino del túnel por el que quiere que salga Probador.

4. Si Probador ha entrado por el camino que le ha gritado Verifier que salga simplemente camina por este camino, si no, Probador susurra las palabras mágicas y sale por el camino que le ha pedido Verifier.

5. Verifier observa salir a Probador por la puerta correcta y ambos salen de la cueva y repiten el proceso hasta que Probador quede convencido.

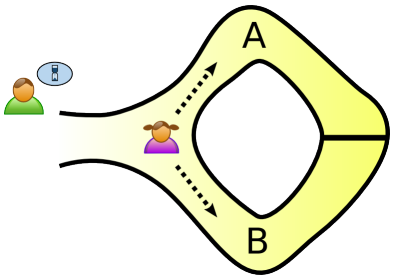

Figura H.1: Probador entra por un camino hasta el fondo mientras Verificador espera fuera(Fuente: *www.wikipedia.org*)

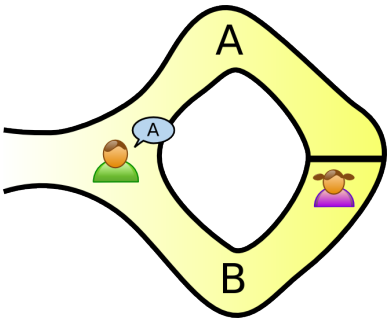

Figura H.2: Verificador entra hasta la bifurcación y elige un camino(Fuente: *www.wikipedia.org*)

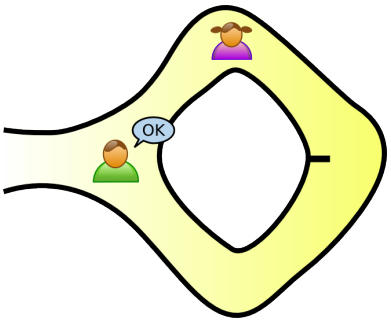

Figura H.3: Probador usa su conocimiento para satisfacer la pregunta(Fuente: *www.wikipedia.org*)

Vemos en el paso 4 que si Probador está en el camino que le pide Verificador no necesita pronunciar las palabras mágicas, es decir, que si Probador fuese un impostor y tuviese suerte, podría engañar a Verificador en una ejecución de la prueba, sin embargo la probabilidad de que esto ocurra, P(Probador entra por la puerta que pide Verificador posteriormente)=$\frac{1}{2}$. La fuerza de las pruebas de conocimiento zero es que si Probador es honesto, podrá salir el 100 % de las veces por la salida del tunel correcta, en este caso, si Verificador es muy escéptico, le basta con resolver $\frac{1}{2^n} < 10^{-50} \Leftrightarrow n > 166$. Por lo que si Verificador y Probador realizan este test 167 veces Verificador puede estar convencido de que Probador es honesto ya que la probabilidad de que un impostor haya satisfecho todos estos tests se considera matemáticamente imposible.

I

Prueba de Conocimiento Cero del Logaritmo Discreto

Después de estudiar el por qué el logaritmo discreto es importante en la criptografía, veamos que existen pruebas de conocimiento cero sobre el logaritmo discreto de un número respecto a una base en un grupo cíclico finito. Como se explica en [36] existe un modo de demostrar que se posee de la solución de un problema de logaritmo discreto sin dar la solución ni ninguna información a partir de la cual se pueda inferir la solución al problema del logaritmo discreto. Establezcamos las bases de este algoritmo:

- A quiere probar a B que conoce la solución x de $\alpha^x = \beta \mod n$ sin revelar información a B.
- A y B acuerdan de antemano el $\alpha\beta$ y n.
- B elige un parÃąmetro $T \in \mathbb{N}$.

Este parámetro T servirá para establecer el nivel de confianza de B en A, ya que la probabilidad a priori que posee B de que A esté falseando la prueba dando resultado positivo y por tanto pudiese estar engaÃśando a B es de 2^{-T}. El protocolo sería el siguiente. Comprobar $log\alpha^x = \beta$ mód n Repetir T veces:

1. A elige de forma aleatoria $r \in_R 1, 2, ..., \phi(n)$.
2. A computa $\gamma = \alpha^r$.
3. A envía a B γ.
4. B elige de forma aleatoria $b \in_R \{0, 1\}$.
5. B envía a A b.
6. A computa $y = r + b * x \mod \phi(n)$.
7. A envía a B y.
8. B chequea $\alpha^y = \gamma\beta^b$.

Veamos que es lo que chequea B:
$\alpha^y = \alpha^{r+b*x} = \alpha^r * \alpha^{b*x} = \alpha^r * (\alpha^x)^b$
$\gamma\beta^b = \alpha^r * \beta^b$
Y vemos que si esto es cierto si A tiene buenas intenciones ya que justo $\beta = \alpha^x \mod n$. El objetivo de el b generado por B de forma aleatoria es evitar los engaños por parte de A.

Está claro que esto es una prueba de conocimiento, ya que se demuestra el conocimiento de la solución al logaritmo discreto. Pero ¿Puede el verificador, en este caso B, inferir algo sobre la solución? La respuesta es que no. Y esto es gracias a que gracias a la suposición equivalente a la de Strong RSA que se encuentrá en el Anexo G.3. Porque no es capaz de resolver el problema del logaritmo discreto.

Esta prueba de conocimiento cero es importante ya que es utilizada en algunos protocolos de firmas grupales.

J

Protocolo de Votado empleando Firmas Ciegas

Este esquema es un esquema similar que se basa en las mismas ideas que el esquema de votación electrónica de Okamoto [24]. Este fue uno de los primeros procesos que realicé de análisis de esquemas de votación electrónica y así como no es el protocolo final desarrollado y estudiado no pertenece a la parte principal del trabajo. Sin embargo si que ha supuesto un proceso de investigación por mi parte este protocolo así como un estudio de su privacidad y seguridad y considero que este trabajo ha formado parte de mi formación inicial en la votación electrónica y por tanto es pertinente que esté presente.

J.0.1. Firmas Ciegas Aplicadas al Voto Electrónico

Como hemos explicado anteriormente las firmas ciegas suponen una posible solución al voto electrónico. La fuerza de las firmas ciegas reside en que una persona puede recibir una firma digital de un documento sin que el firmante pueda obtener información sobre lo que está firmando. De este modo y de forma muy simplificada si conseguimos que un ciudadano envíe su voto ofuscado y la administración le firma este voto ofuscado y luego el votante es capaz de extraer la firma y enviar su voto con la firma de forma anónima se consigue anonimato, además puesto que este voto está ofuscado el que la administración sepa a quién se le expide una firma no compromete la seguridad ni anonimato del sistema. Veamos más en detalle como funcionaría este protocolo.

- Se confía en la administración para la realización del conteo de votos

- Todos los votantes confían en la autoridad certificadora que emite el certificado digital de la administración A y la administración confía en la autoridad certificadora que emite los certificados digitales de los votantes

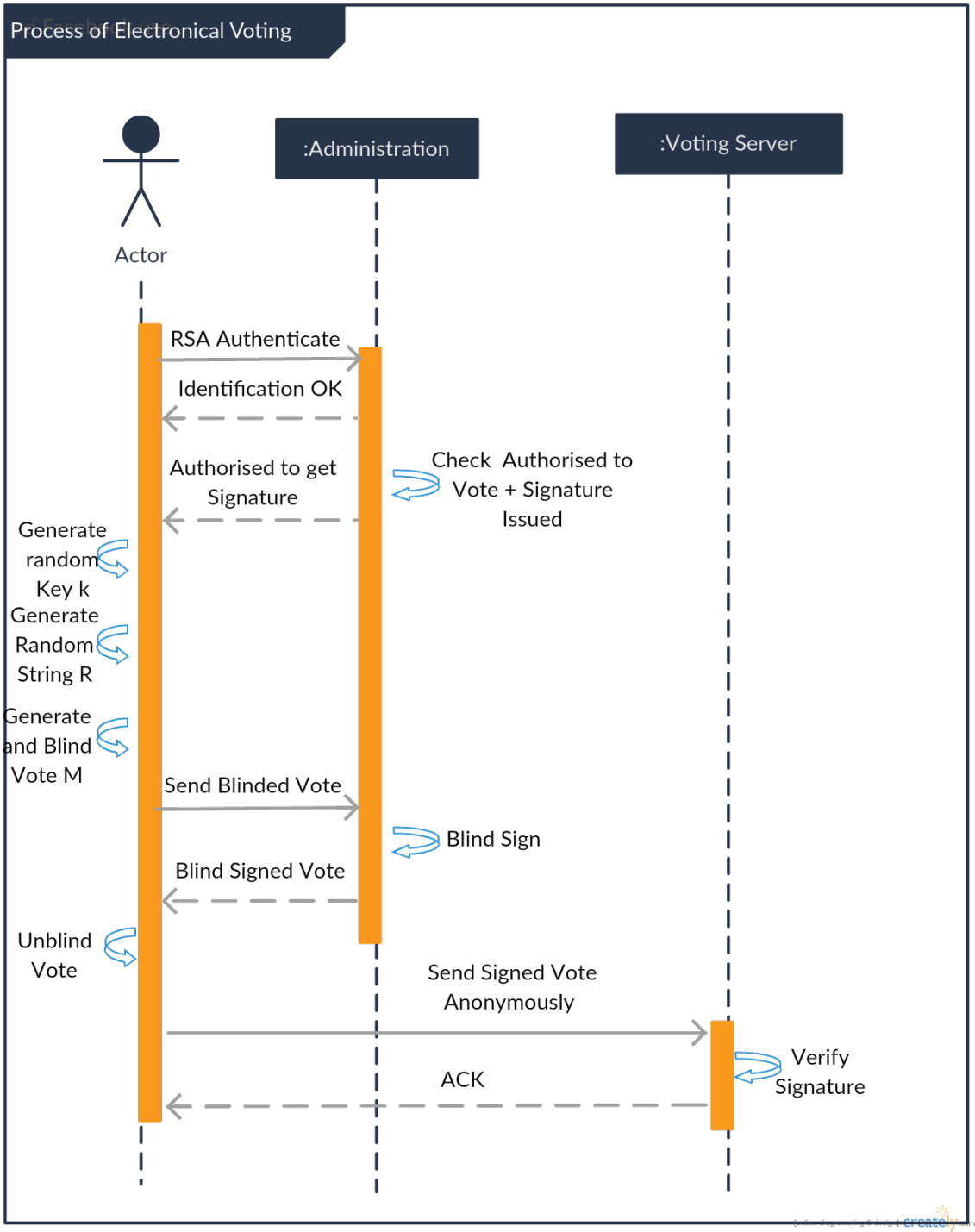

Figura J.1: Voto electrónico usando Firmas Ciegas

Un posible protocolo de votado usando firmas ciegas es el siguiente:

1. Se eligen de antemano S_A y S'_A de forma privada y una función de cifrado simétrico C y C' tal que $C^{-1}(k, S_A(C(k, x))) = S_A(x)$ donde S_A es la función de cifrado con la clave privada de la administración A y S'_A la función de cifrado con clave pública. Esta k solo la conocerá el cliente que será el que la elija.

2. Cada votante elige una clave k de forma aleatoria del espacio de claves de C y una cadena aleatoria R y genera su mensaje de voto $M = x||Hash(x||R)$. El objetivo de R es poder

identificar a posteriori el voto del votante por parte del votante y de nadie más.

3. El votante establece su deseo de votar a la administración mediante su certificado digital y el protocolo RSA.
4. La administración verifica que el ciudadano está autorizado a votar y que no se le ha expedido una firma antes en la tabla que guarda apropiadamente para esto.
5. Si el votante está autorizado a votar y no se le ha expedido una firma, que será lo que le permita votar(esta firma expedida es análoga a la firma expedida por el comisario explicada en las firmas ciegas Firmas Ciegas), recibe el visto bueno del comisario, el votante genera una clave k de forma aleatoria del espacio de claves de C, computa $C(k, M)$ donde M es su voto y se lo envía al comisario
6. El comisario recibe $C(k, M)$ y computa $S_A(C(k, M))$ y se lo envía al votante.
7. El votante recibe $S_A(C(k, M))$ y computa $S_A(M)$ mediante $C^{-1}(k, S_A(C(k, M))) = S_A(M)$.
8. El votante ya tiene su voto firmado por la autoridad siendo este $M||S_A(M)$.
9. El votante envía de forma anónima $M||S_A(M)$ al comisario. Este envío de forma anónima podría realizarse mediante el uso de redes anonimizadoras como puede ser TOR TOR(The Onion Route).
10. La administración comprueba la validez del mensaje comprobando que este ha sido firmado por él realizando la comprobación $S'_A(S_A(M)) = M$.
11. Si el voto pasa la verificación de la administración lo almacena como voto válido.
12. Al final de la jornada electoral el comisario publica la lista de mensajes M en una lista accesible por todos los votantes.
13. Cada votante puede generar de nuevo M puesto que cada uno conoce su voto x y su cadena aleatoria R y busca en la lista de votos su correspondiente M, esto le garantiza que su voto se ha contado y que no ha sido modificado.

Ahora que está explicado este esquema de voto veamos las condiciones deseables del voto electrónico y cómo se cumplen con este esquema

- Solo pueden votar los votantes autorizados para ello, pues un votante tiene que identificarse mediante su certificado digital al servidor, de este modo si el servidor posee un listado de autorizados para votar y consideramos la igualdad Certificado Digital = Persona no se podrá votar a no ser que no se tenga autorización para ello
- Una persona solo puede votar como máximo una vez, esto se consigue mediante el control de la expedición de firma digital
- La identidad de una persona no puede ser suplantada ya que la identificación es unívoca mediante su certificado digital
- El voto es secreto, ya que la administración no puede extraer el voto de un usuario cuando pide expedición de firma. Y cuando el usuario entrega el voto de forma anónima no entrega además ningún tipo de información suya
- El voto no es manipulable ni modificable, y en este caso no lo es pues el usuario entrega una firma de su voto, una firma digital hecha por la administración junto al mismo, y por tanto el voto no se puede modificar sin detección

- El conteo de votos debe de ser preciso, y este lo es de forma trivial ya que el conteo es binario y muy rápido, o se cuenta el voto o no se cuenta.

- El proceso de voto debe ser auditable, y este lo es porque se puede comprobar quien ha votado y quien no y en qué momento. Sin embargo no se puede extraer el voto de ningún votante ni siquiera bajo demanda. El usuario puede demostrar que su voto por ejemplo no se ha contado recurriendo a la cadena aleatoria generada.

Análisis de Seguridad del Esquema En esta sección se hará un análisis de seguridad de este esquema de votado

En este esquema se parte de la base de que todos los usuarios tienen ya su clave pública y privada y que son capaces de establecer una comunicación RSA autenticándose.

En todas las comunicaciones que se hagan de forma anónima a cualquier servidor se utilizará la clave pública de este servidor mediante TLS/SSL para asegurarse que el mensaje solo puede ser leído por el servidor destino y otorgar un test que asegure la integridad del mensaje en esta comunicación, como viene explicado en el anexo TLS/SSL garantizamos que esto funcione de forma correcta y la comunicación TLS/SSL es suficientemente segura.

Dado que aquí un usuario puede pedir una firma de un voto hay que verificar que a un usuario solo se le expida una firma de un voto, es por esto que el proceso de verificación de que no se le ha expedido firma a un votante tiene que estar protegido mediante el control de semáforos ya que de no ser así un usuario podría iniciar N conexiones paralelas a la vez y si no está protegida la comprobación podría conseguir N votos firmados válidos en vez de solo 1.

Un mal control de flujo sobre los datos guardados por los servidores puede suponer un agujero de seguridad en el mismo. Por lo que debemos asegurarnos en bloquear los recursos antes de escribir, y que el proceso de lectura sea bloqueante del mismo modo que lo es la escritura, de este modo si lo que hacemos es hacer un bloqueo, leer, verificar que no se le ha emitido firma, escribir, y acto seguido liberar el recurso solucionamos este problema.

Análisis de Ciberataques al Sistema El análisis de ciberataques más comunes a este protocolo son análogos a los del protocolo que utiliza firmas grupales Análisis de Ciberataques al Sistema

Pérdidas de la conexión Del mismo modo que en el protocolo anterior vamos a estudiar que consecuencias puede tener la pérdida de la conexión en medio de la comunicación del protocolo de votado por parte de un cliente

- Si el usuario pierde la conexión antes de que se le haya expedido la firma pero se le haya identificado como miembro válido para votar, puede volver a iniciar la conexión y el protocolo.

- Si el usuario ha perdido la conexión después de enviar su voto cegado a la administración pero antes de recibirlo firmado al identificarse de nuevo en el servidor podrá recuperar este voto, y podrá recuperarlo pues ha guardado la K de la función de cegado.

- Si el usuario pierde la conexión antes de recibir la confirmación del guardado de su voto, cuando lo vuelva a emitir el servidor le responderá que ese voto ya existía en el sistema.

Auditoría del Voto A diferencia de el esquema de votado anterior basado en firmas grupales, este esquema de voto no cuenta con una auditoría del 100 % del proceso de votado, sin embargo cuenta con una auditoría con la que cuentan muchos sistemas de voto actuales, mediante un resguardo de voto, que puede ser implementado mediante un Hash, el votante puede comprobar que su voto se ha guardado de modo correcto. Pero esta auditoría es más bien una comprobación ya que un votante no podrá demostrar la autoría de un voto mediante una ZKP como puede hacer mediante firmas grupales.

Análisis de Anonimato del Voto En esta sección estudiaremos qué información se intercambia en qué momentos para ver que se preserva el anonimato del voto y comprobar que la administración no aprende nada del votante.

- Cuando el votante se comunica con la administración, la administración conoce la identidad del votante ya que este se ha tenido que autenticar mediante su clave RSA, lo que el comisario no puede hacer es ver lo que está firmando digitalmente ya que a pesar de conocer el algoritmo de cifrado simétrico utilizado por el votante, no es capaz de probar en un tiempo computacionalmente viable la clave utilizada por el votante. Por lo que no gana ninguna información del voto.
- Cuando el votante envía su voto al servidor de voto, este lo único que puede comprobar del voto es que es un voto firmado correctamente, pero al ser enviado de forma anónima el votante no envía ningún tipo de información junto al voto, es más, en el caso de usar una comunicación anónimizadora como TOR o una VPN ni la dirección IP del voto propicia ningún tipo de información.

K

SQL Injection

La técnica SQL Injection se conoce como la práctica de otorgar datos malintencionados a una página web o a un servicio web con intención de, si el sistema no está protegido contra estos ataques, obtener información de la base de datos, eliminar datos específicos, o insertar datos a placer.

Para ilustrar esto vamos a partir de un ejemplo sencillo pero ilustrativo de como se podría hacer SQL Injection a una página web no protegida. Supongamos que existe un servicio de login web de una página en la que se nos pide usuario y contraseña, supongamos también que esta página la forma que tiene de comprobar usuario y contraseña es la siguiente:

```
1 SELECT COUNT(*)
2 FROM Login
3 WHERE User=%s AND Password=Hash(%s);
```

Una consulta muy inocente y sencilla que comprueba el usuario y el hash de la contraseña de un usuario, posteriormente si esta consulta le devuelve un 1, el login será válido, y si le devuelve un 0, el login será no válido.

Supongamos que un usuario malintencionado averigua, o intuye que la tabla de login de usuarios se llama login, Login, Users, etcétera, cosa que resulta muy natural, y supongamos que para cada término de estos, llamemos a cada termino en cuestión $Tabla_i$ y envía como usuario lo siguiente:

```
1 foo;DROP Table $Tabla_i$; #
```

Al ejecutar la sentencia quedaría algo así:

```
1 SELECT COUNT(*)
2 FROM Login
3 WHERE User=foo;DROP Table $Tabla_i$;# AND Password=Hash(%s);
```

Donde todo lo que sigue a # es texto comentado y no será ejecutado, de este modo un atacante habría logrado eliminar todos los usuarios válidos de la base de datos utilizando unos pocos minutos de su tiempo resultando en un ataque DOS instantáneo.

Por supuesto este es solo un tipo de ataque de SQL Injection, si fuese de una página en la que, hacemos una búsqueda, y la página escribe todos los resultados de respuesta de la base de datos las consecuencias pueden ser incluso mayores. Supongamos que estamos visitando una página de agencia de viajes que te permite filtrar por destino, precio, fechas, etcétera. Supongamos que la consulta tiene esta estructura:

```
1 SELECT *
2 FROM Hotel
3 WHERE $Restricciones$;
```

Supongamos que el usuario malintencionado intuye que lo que está haciendo la página es consultar a una base de datos e imprimir lo que le devuelva. Si el usuario envía una consulta de este estilo en, supongamos, hotel.

```
1 ' AND 1=0 UNION (Cualquier sentencia SQL); #
```

Lo que consigue es que, para empezar la primera sentencia habitual no de resultados pues 1 siempre es distinto de 0. Y después concatena un resultado vacío con la consulta que le plazca, aquí el usuario malintencionado puede volcarse la base de datos completa a la que tenga acceso el sitio web.

Tambien existen los ataques por Blind SQL Injection es otro tipo de ataque que se basa en que el sistema solo muestra resultados cuando el resultado es correcto y cuando es erróneo no. Determinando de cierto modo la estructura de la sentencia SQL mediante pruebas viendo cuando la sentencia introducida de modo malicioso se ejecuta de modo correcto o no.

K.1. Medidas de seguridad

Una buena medida de seguridad inmediata ante el SQL Injection es la de escapar todos los caracteres que no se espera que un usuario introduzca, es decir, si sabemos de antemano que el login de un usuario solo admite letras y números, escapar todo lo demás. En general escapar todos los caracteres para que sean tratados como cadenas, que es lo que son. De este modo, un ataque por SQL Injection quedaría mitigado ya que no ejecutarías sentencias del atacante sino resultaría en ejecuciones con sentencias con campos que contienen sentencias, que potencialmente fallarían.

No obstante no es la única medida en absoluto, también hay que vigilar que permisos tiene que proceso, que un sitio web, por ejemplo, solo pueda acceder a las tablas que necesite, que no tenga permisos de root, establecer claramente los permisos de lectura escritura que tiene cada usuario y sobre que tablas, para evitar que, si hubiese una vulnerabilidad, tuviese el menor impacto posible sobre el sistema.

L

DoS/DDoS

Los ataques DoS son unos de los más comunes utilizados hoy día, la idea es muy sencilla, consiste en impedir, dificultar, o ralentizar las comunicaciones entre unos clientes legítimos y un servicio. Para realizar este tipo de ataques se puede operar de muchas maneras, desde hacer volar el tendido eléctrico de la fuente de alimentación del servicio, a intentar hackear una base de datos y corromper datos de login a una red social por ejemplo, o la forma más popular y con menos consecuencias legales, estos últimos se basan en hacer una cantidad desorbitada de peticiones "legítimas.ª un servicio de modo que el servicio esté ocupado la mayor porción del tiempo en atender estas peticiones que las de los otros usuarios que quieren utilizarlo. De hecho si un ataque DOS consigue saturar la pila de peticiones de un servicio no es que consiga que la petición de el resto de usuarios tarde mucho en hacerse, sino que consigue que la petición ni se llegue a procesar. Este tipo de ataques se suelen realizar generalmente desde un equipo, para ello existen ya herramientas desarrolladas con este propósito como LOIC(Low Orbit Ion Cannon) o HULK.

Una variante más sofisticada que el ataque DoS es el ataque DDoS, que es Distributed Denial of Service. El motivo de existencia de este ataque es que el DoS es fácilmente resoluble ya que con un firewall muy básico puede bloquear una dirección IP por recibir una tasa de peticiones que se considera excesiva. El DDoS como su nombre indica, es distribuido, y es más difícil de resolver, pues en DoS si queríamos hacer X peticiones/segundo si este X era muy elevado era fácil detectar el intento de ataque, sea el umbral de DOS de un firewall Y peticiones/segundo, si ponemos N ordenadores a realizar el ataque cada uno a una tasa de X/N peticiones segundo de tal modo que $X/N < Y$ conseguimos el mismo resultado y una tarea más difícil de resolver al servidor en cuestión.

Dado lo difícil que es utilizar tantos ordenadores al mismo tiempo por parte de un usuario para realizar estos ataques se suelen utilizar botnets que son redes de ordenadores infectados por un pequeÃśo virus informático que da control en cierto grado a un usuario remoto sobre este equipo. De este modo un usuario malintencionado es capaz de infectar cientos, miles de ordenadores de gente que luego serán utilizados para el interés de esta persona. Dado que estos virus se limitan a hacer unas cuantas peticiones bajo demanda del artífice son altamente complicados de detectar. Un ejemplo de este ataque viene reflejado en la figura D.1.

Los ataques DOS son polémicos cuanto menos ya que se han llegado a plantear y hasta considerar en algunos paises como un modo legítimo de protesta.

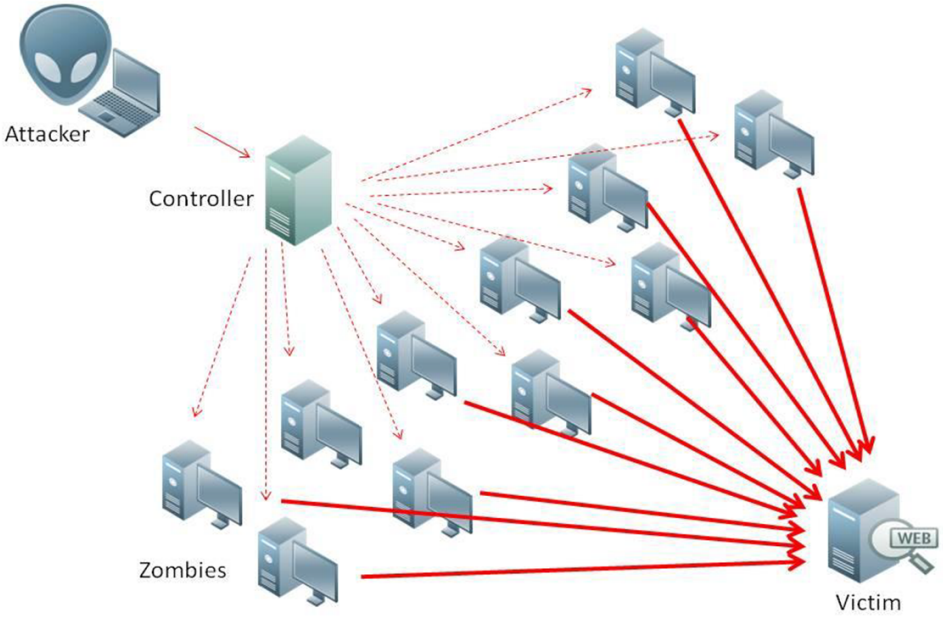

Figura L.1: Ataque DDoS (Fuente: *www.wikipedia.org*)

M

Man in Middle

El ataque por Man in the Middle, ataque del Hombre en el Medio es un ataque muy auto-explicativo, consiste en que el atacante consigue redirigir el trÃąfico entre dos equipos para que pasen por él para poder husmear entre los paquetes que circulan, y ademÃąs de poder ver en ciertos casos estos paquetes, puede limitar que trÃąfico se envía, puede intentar hacerse pasar por el cliente o servidor, y derivar en multitud de ataques.

¿Cómo se realizan los ataques Man in the Middle? Un ataque Man in the Middle puede ser fÃącilmente realizable dentro de una red local mediante el ARP poisoning, sea IP_A y MAC_A la dirección IP y la MAC de la interfaz de red de ese equipo en una determinada red. Sea IP_B y MAC_B la de otro equipo con quien se quiera comunicar A, sea en este caso B por ejemplo el router de la red. y sea MAC_M la MAC del atacante. Si el atacante genera paquetes ARP de la siguiente forma:
IP=IP_A is at MAC=MAC_M
IP=IP_B is at MAC=MAC_M
Y envía estos paquetes de forma relativamente constante, las cachés de ambos equipos quedarÃąn corruptas y todos los paquetes de la conexión legítima pasarÃąn por el atacante.

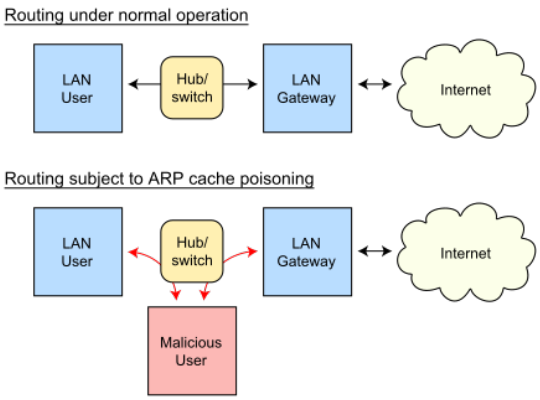

Figura M.1: Ataque Man in the Middle(Fuente: *www.wikipedia.org*)

M.1. Explotación

Una vez el ataque MitM se ha realizado de forma satisfactoria, ¿Qué podría hacer un atacante? Lo primero y más inmediato es un ataque DoS, negar el tráfico y dejar sin conexión a el cliente, pero esto es lo mínimo que puede hacer el atacante, un atacante puede husmear todo el tráfico que no vaya encriptado. De hecho un atacante puede intentar filtrar el tráfico de modo que fuerce al cliente pedir sus conexiones en HTTP y no en HTTPS para poder ver todo su contenido web.

Otro ataque conocido y muy potente realizado a través de Man in the Middle es el de estar en el medio también para las conexiones cifradas, esto se consigue en el handshake, es decir, cuando el servidor envía su certificado digital, el atacante genera su propio certificado digital y se lo envía al cliente, el cliente utiliza este certificado para mandar sus mensajes por lo que el atacante es capaz de descifrarlo, y con el certificado auténtico del servidor que él ha recibido puede enviar los mensajes como si fuese él el propio cliente.

M.2. Prevenciones

La prevención más inmediata es la de no conectarse a redes no conocidas o redes que no sean de confianza, y de hacerlo no manejar nunca información sensible ni utilizar conexiones a servicios que nos sean de importancia. Como medidas proactivas existen herramientas como ARPcop que monitoriza en tiempo real entradas duplicadas en la tabla ARP del equipo y notifica al usuario de estos sucesos. Además se recomienda utilizar siempre conexiones cifradas. Y para evitar el último tipo de ataque que se basa en creación dinámica de certificados como medida es aceptar solo certificados firmados por autoridades certificadoras de confianza y rechazar los certificados autofirmados, que son generalmente los certificados creados por los atacantes para este tipo de ataques.

N

TOR(The Onion Route)

The Onion Router, o más bien conocido como TOR, es una red superpuesta a internet cuyo objetivo es garantizar en la medida de lo posible un uso anónimo de internet. Un modo viene explicado en [35].

N.1. Funcionamiento

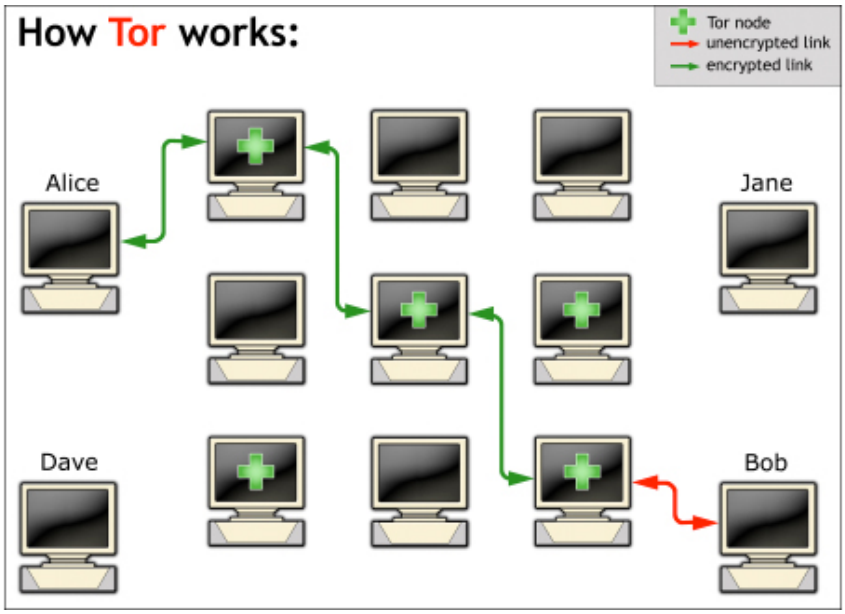

Figura N.1: Ejemplo de comunicación de la red TOR (Fuente: *www.wikipedia.org*)

TOR es una red de equipos que están distribuidos a lo largo del mundo, estos equipos tienen como objetivo prestar el servicio de navegación mediante TOR, ¿Cómo funciona esto? Un cliente se tiene que comunicar con un nodo de esta red para solicitar el servicio, y luego es este equipo que acepta la conexión, en colaboración con los otros equipos los que garantizan este servicio,

¿Cómo se garantiza? Primero el cliente debe estar usando un medio apropiado de conexión, ya sea mediante el uso del servicio TOR, que es básicamente un puerto de un equipo que transmite todo paquete recibido mediante TOR, o un navegador apropiado, este servicio o navegador sabe, y no vamos a centrarnos en cómo lo sabe, que nodos de TOR están disponibles y son deseables, i.e. un nodo relativamente cercano al cliente, una vez el cliente conoce a qué servidor se tiene que conectar, el servicio es capaz de obtener una clave pública de cifrado del servidor nodo entrante, de este modo todo lo que envíe a TOR lo hará cifrándolo con esta clave. Una vez el cliente ya tiene la clave, envía su paquete al nodo entrante, de este modo ni los ISP ni la policía ni ninguna autoridad pertinente es capaz de saber a priori ni si quiera que servicio está solicitando a través de TOR. Una vez el nodo entrante tiene el paquete selecciona mediante un criterio el siguiente nodo de la conexión, y se lo envía de modo cifrado, y este nuevo nodo repite el proceso, así N veces, y al finalizar el nodo saliente realiza la petición en cuestión al servicio que solicita el cliente, y la respuesta se recibe por el mismo circuito que se ha establecido. De este modo el servicio que solicita el cliente queda oculto. Además el nodo que solicita la petición por el cliente está potencialmente muy alejado del propio cliente lo cual significa que en gran medida es muy difícil asociar que la petición viene de el cliente en cuestión.

N.2. Anonimato

Utilizando este servicio queda claro que a priori no se puede descifrar ni saber de donde viene el paquete de este cliente. Sin embargo un uso reiterado del servicio y del circuito podrían llevar a realizar un análisis de tráfico y facilitar un acotamiento de donde vienen los paquetes, es por esto que TOR cada cierto número de peticiones reinicia el circuito generando uno completamente nuevo y diferente al anterior, siendo esto que utilice nodos de otros paises por ejemplo.. Además el tráfico de acceso a TOR va encriptado por lo que no se puede saber que servicio que solicita el paquete en cuestión. La única información que se puede inferir a partir de una conexión a TOR es que la conexión en cuestión está utilizando TOR, y nada más, es por esto que TOR no suele contar con el beneplácito de gobiernos y organismos de seguridad.

N.3. Servicios Ocultos

Uno de los motivos de la fama de TOR es que presta servicios ocultos al resto de internet. Gracias a que posee esta red superpuesta a Internet en la que todo el tráfico va cifrado, ofrece servicios ocultos en equipos que fuera de internet no se conocen. Estos servicios solo se conocen dentro de TOR, y son URLs personalizadas acabadas en .onion. No vamos a entrar en cómo funciona el dar de alta un servicio oculto en TOR pero se hace de modo que, todas las URLs onion resultan accesibles desde los nodos de tor, y suelen ser extremadamente difíciles de rastrear y determinar dónde se encuentra el servidor en cuestión, es por esto que TOR, a pesar de ser concebido como algo bueno, acaba siendo utilizado también para cosas malas, ya que entre los servicios ocultos se ocultan páginas de tráfico de drogas, tráfico de armas, tráfico de personas, pornografía infantil, servicios de asesinos a sueldo y un largo etcétera de actividades delictivas.

Ñ

Medidas de tiempos de esquemas de firmas grupales de libgroupsig

En este anexo se muestran una serie de medidas tomadas de la librería libgroupsig respecto a los esquemas de firmas grupales que incorpora. Todas estas gráficas han sido sacadas del artículo [1].

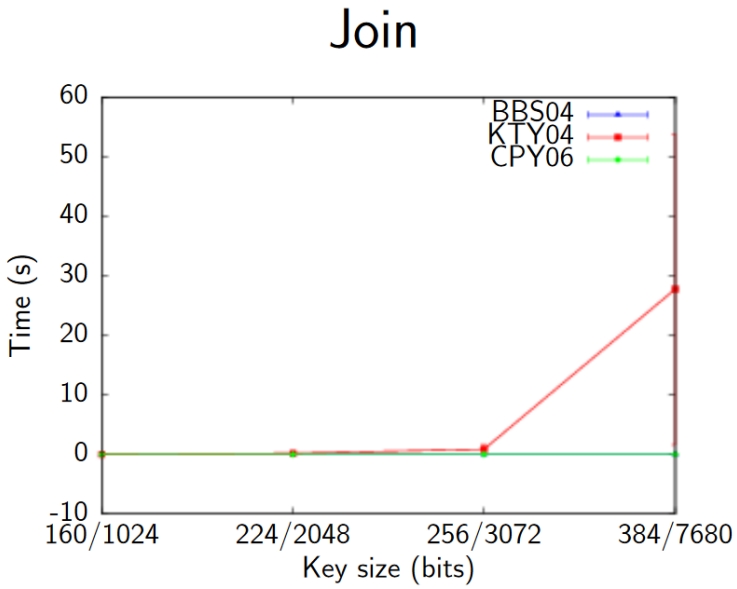

Figura Ñ.1: Medidas de tiempo de Join

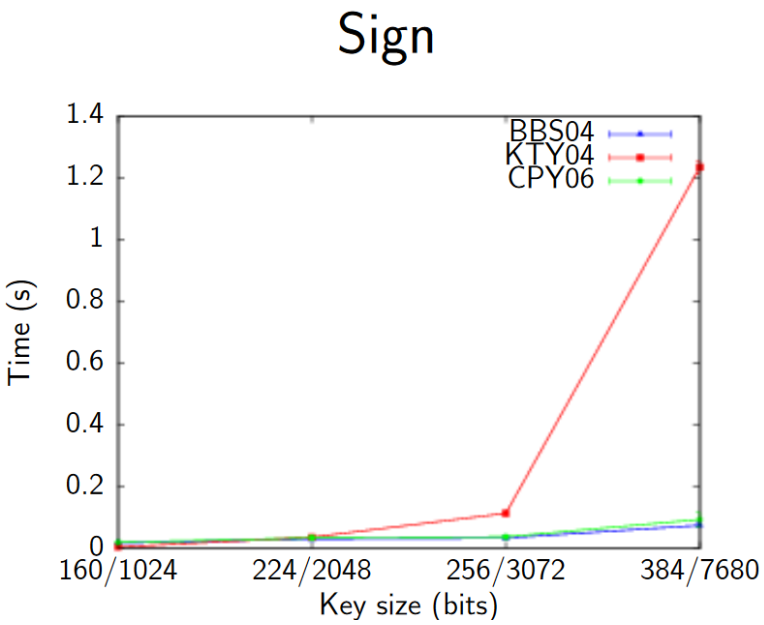

Figura Ñ.2: Medidas de tiempo de Sign

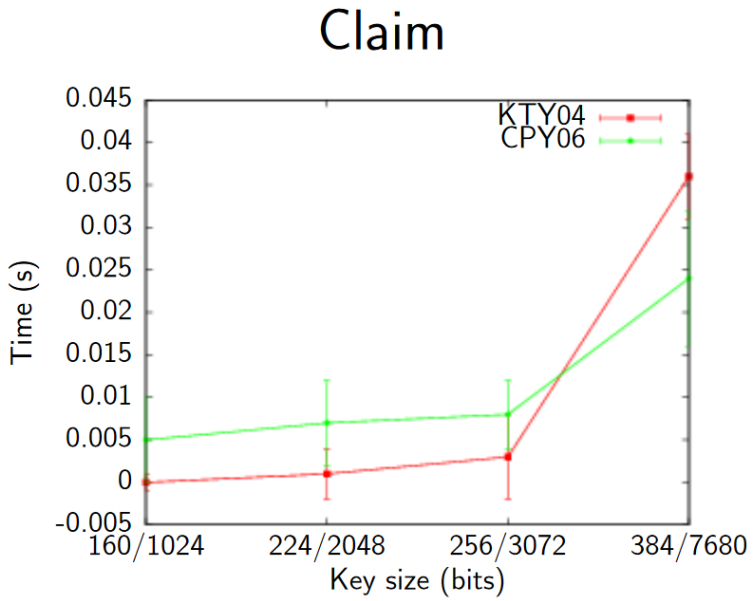

Figura Ñ.3: Medidas de tiempo de Claim

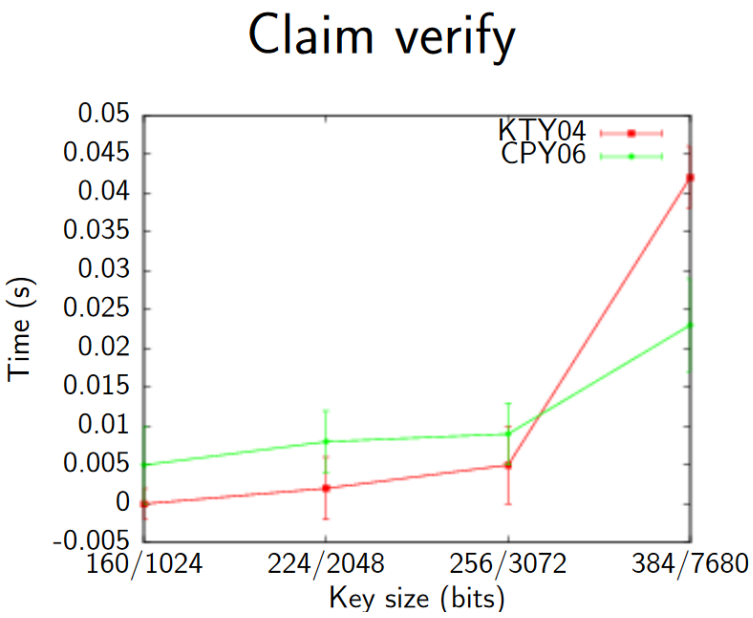

Figura Ñ.4: Medidas de tiempo de Claim_Verify

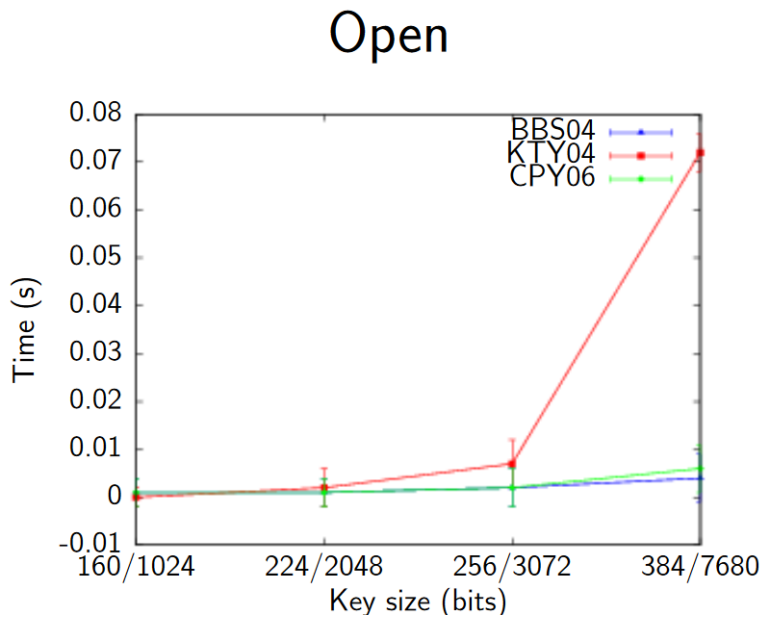

Figura Ñ.5: Medidas de tiempo de Open

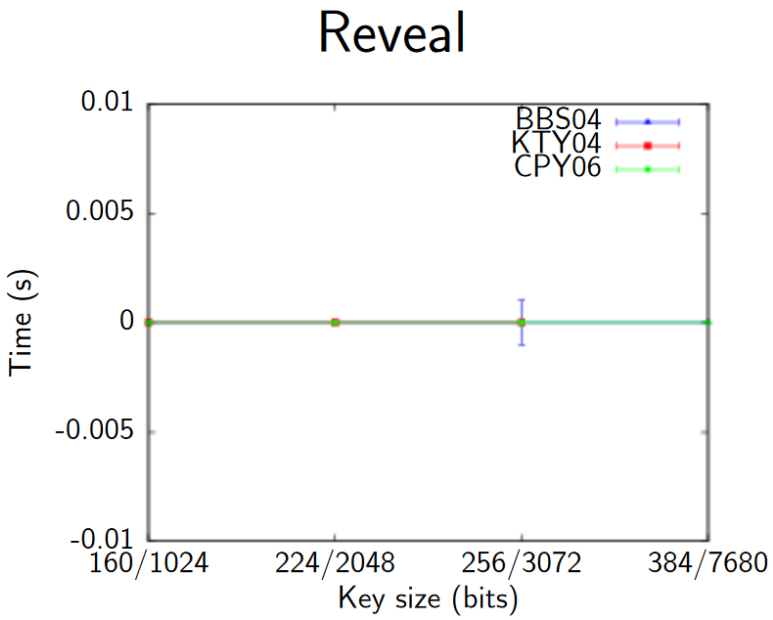

Figura Ñ.6: Medidas de tiempo de Reveal

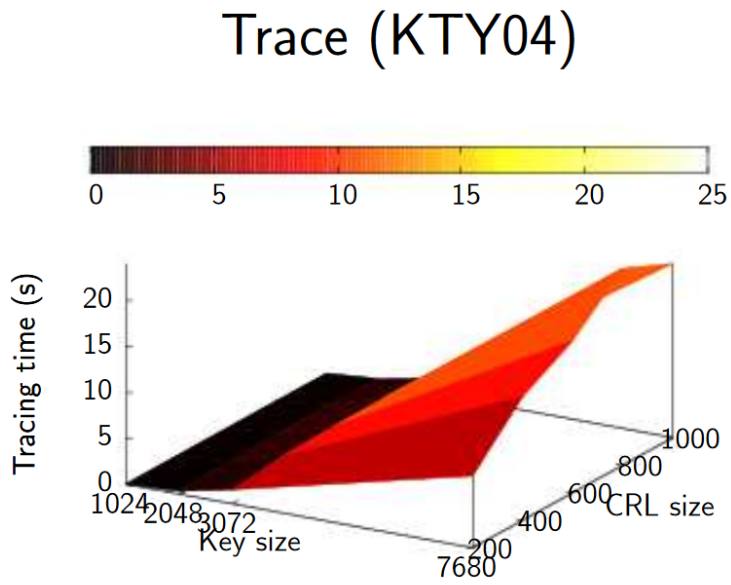

Figura Ñ.7: Medidas de tiempo de Trace_KTY04

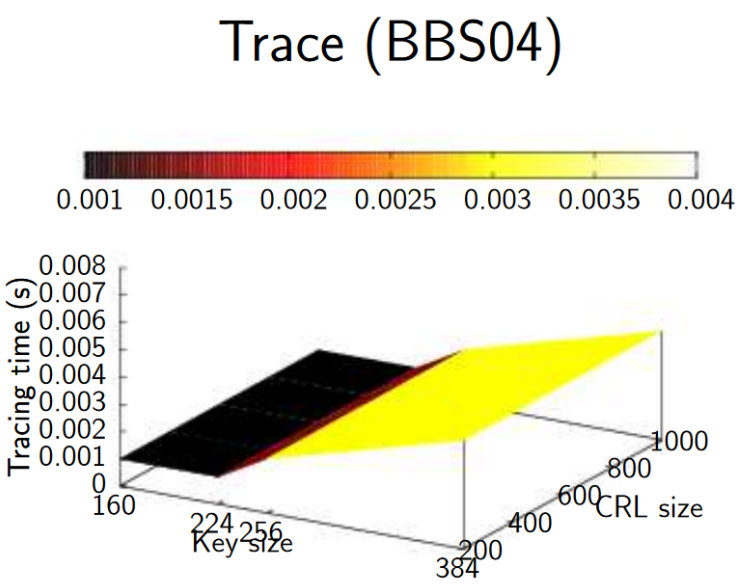

Figura Ñ.8: Medidas de tiempo de Trace_BBS04

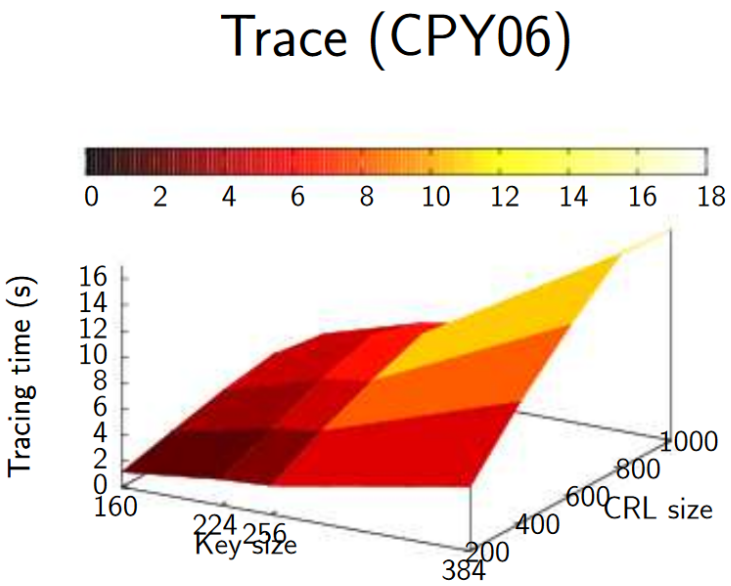

Figura Ñ.9: Medidas de tiempo de Trace_CPY06

Como se puede apreciar en las gráficas, a medida que vamos aumentando el tamaño KTY04 llega un momento que sufre un crecimiento significativamente mayor que los otros esquemas BBS04 y KTY04, esto es dado a que la tecnología intrínseca a KTY04 es la de RSA, mientras que en los otros esquemas es utilizando curvas elípticas, consiguiendo un rendimiento mayor y una seguridad equivalente con claves significativamente más pequeñas.

O Código

O.1. Código de Cliente

```
import os.path
import protocol_codes
import sys
import socket
import ssl
import subprocess
from Crypto.PublicKey import RSA
size = 4096
class VotingClient(object):
    def __init__(self, host_g1, port_g1, host_A, port_A, host_v, port_v,
        member_key_file, group_key_file):
        self.member_key_file = member_key_file
        self.group_key_file = group_key_file
        self.host_g1 = host_g1
        self.port_g1 = port_g1
        self.host_a = host_a
        self.port_a = port_a
        self.host_v = host_v
        self.port_v = port_v
        self.tokenPS = None
        self.tokenPS_file = "tokenPS_file.txt"
        self.tokenPS_sig = None
        self.tokenPS_sig_file = "tokenPS_sig_file.txt"
        self.tokenPS_sig_proof = None
        self.tokenPS_sig_proof_file = "tokenPS_sig_proof_file.txt"
        self.tokenV = None
        self.tokenV_file = "tokenV_file.txt"
        self.vote_proof_file = "vote_proof_file.txt"
        self.composed_vote_file = "composed_vote_file.txt"
        self.signed_composed_vote_file = "signed_composed_vote_file.txt"
        self.proof_signed_composed_vote_file = "
            proof_signed_composed_vote_file.txt"
        self.tokenPS_TokenV_file = "tokenPS_TokenV_file.txt"
        self.tokenPS_TokenV_sig_file = "tokenPS_TokenV_sig_file.txt"
        self.tokenPS_TokenV_sig_proof_file = "tokenPS_TokenV_sig_proof_file.
            txt"
        self.sock_g1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
35          self.tls_sock_g1 = ssl.wrap_socket(self.sock_g1, ssl_version=ssl.
                PROTOCOL_TLSv1, cert_reqs=ssl.CERT_NONE)
36          self.sock_a = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
37          self.tls_sock_a = ssl.wrap_socket(self.sock_a, ssl_version=ssl.
                PROTOCOL_TLSv1, cert_reqs=ssl.CERT_NONE)
38          self.sock_v = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
39          self.tls_sock_v = ssl.wrap_socket(self.sock_v, ssl_version=ssl.
                PROTOCOL_TLSv1, cert_reqs=ssl.CERT_NONE)
40
41      def group_sign_document(self, sig_file, input_file):
42          p = subprocess.Popen(['./tools/sign','CPY06',sig_file,input_file,
                self.member_key_file,self.group_key_file])
43          p.wait()
44          return True
45      def claim_signature(self,proof_file,sign_file):
46          p = subprocess.Popen(['./tools/claim','CPY06',self.member_key_file,
                self.group_key_file,proof_file,sign_file])
47          p.wait()
48          return True
49
50      def dump_in_file(self, file_name, content):
51          tmp_file = open(file_name,"w")
52          tmp_file.write(content)
53          tmp_file.close()
54          return
55
56      def get_tokenPS_from_file(self):
57          tmp_file = open(self.tokenPS_file,"r")
58          self.tokenPS = tmp_file.read()
59          tmp_file.close()
60
61      def get_tokenPS_sig_from_file(self):
62          tmp_file = open(self.tokenPS_sig_file,"r")
63          self.tokenPS_sig = tmp_file.read()
64          tmp_file.close()
65
66      def get_tokenPS_sig_proof_from_file(self):
67          tmp_file = open(self.tokenPS_sig_proof_file,"r")
68          self.tokenPS_sig_proof = tmp_file.read()
69          tmp_file.close()
70
71      def get_tokenV_from_file(self):
72          tmp_file = open(self.tokenV_file,"r")
73          self.tokenV = tmp_file.read()
74          tmp_file.close()
75
76      def get_vote_proof_from_file(self):
77          tmp_file = open(self.vote_proof_file,"r")
78          proof = tmp_file.read()
79          tmp_file.close()
80          return proof
81
82
83      def get_file_content(self,file):
84          tmp_file = open(file,"r")
85          content = tmp_file.read()
86          tmp_file.close()
87          return content
88
89
90      def validate_tokenPS(self):
91          if not self.tokenPS:
92              return False
```

```
93          try:
94              self.tls_sock_g1.connect((self.host_g1,self.port_g1))
95          except Exception as error:
96              print "An error has occured"
97              print error
98              sys.exit()
99
100         self.tls_sock_g1.sendall("Validate TokenPS:"+self.tokenPS)
101         response = self.tls_sock_g1.recv(4096)
102         if "VALID" in response:
103             return True
104         elif "WRONG" in response:
105             return False
106     def get_new_tokenPS(self):
107         try:
108             self.tls_sock_g1.connect((self.host_g1,self.port_g1))
109         except Exception as error:
110             print "An error has occured: A1"
111             print error
112             sys.exit()
113
114
115         self.tls_sock_g1.sendall(protocol_codes.request_tokenPS)
116
117         response = self.tls_sock_g1.recv(size)
118         if protocol_codes.tokenPS not in response:
119             print "Error claiming the TokenPS"
120             raise Exception('Error','Server answered<'+response+'> to
                TokenPS request')
121         self.tokenPS = response.replace(protocol_codes.tokenPS,'')
122         print "Challenge TokenPS = " + self.tokenPS
123         tmp_file = open(self.tokenPS_file,"w")
124         tmp_file.write(self.tokenPS)
125         tmp_file.close()
126         print "signing tokenPS"
127         self.group_sign_document(self.tokenPS_sig_file, self.tokenPS_file)
128         print "tokenPS signed"
129         tmp_file = open(self.tokenPS_sig_file, "r")
130         self.tokenPS_sign = tmp_file.read()
131         tmp_file.close()
132         print "Signature="+self.tokenPS_sign
133
134
135         self.tls_sock_g1.sendall(self.tokenPS_sign)
136
137         response = self.tls_sock_g1.recv(size)
138
139         if response == protocol_codes.revoked_user:
140             print "TokenPS couldn't be validated because voter has already
                been validated"
141         elif response == protocol_codes.invalid_group_signature:
142             print "Signature is not valid"
143         elif response == protocol_codes.activated_tokenPS:
144             print "Activated TokenPS"
145         self.tls_sock_g1.close()
146         return response
147
148
149
150     def tokenPS_accounting(self,tokenPS,proof):
151         try:
152             self.tls_sock_g1.connect((self.host_g1,self.port_g1))
153         except Exception as error:
```

```
154             print "An error has occured: A"
155             print error
156             sys.exit()
157         self.tls_sock_g1.sendall(protocol_codes.tokenPS_accounting + tokenPS
                + "&" + proof)
158         response = self.tls_sock_g1.recv(size)
159         if response == protocol_codes.accounting_ok:
160             return True
161         else:
162             return False
163
164     def tokenV_accounting(self,TokenV,proof):
165
166         try:
167             self.tls_sock_a.connect((self.host_a,self.port_a))
168         except Exception as error:
169             print error
170             sys.exit()
171
172         self.tls_sock_a.sendall(protocol_codes.tokenV_accounting + TokenV +
                "&" + proof)
173         response = self.tls_sock_a.recv(size)
174
175         return response == protocol_codes.accounting_ok
176
177     def vote_accounting(self,tokenV,vote_proof):
178         try:
179             self.tls_sock_v.connect((self.host_v,self.port_v))
180         except Exception as error:
181             print error
182             sys.exit()
183         query = protocol_codes.vote_accounting + tokenV + "&" + vote_proof
184
185         self.tls_sock_v.sendall(query)
186         response = self.tls_sock_v.recv(size)
187         return response == protocol_codes.accounting_ok
188
189
190
191
192
193     def get_new_tokenV(self):
194         print "Connecting to TokenV dispatcher"
195
196         try:
197             self.tls_sock_a.connect((self.host_a,self.port_a))
198         except Exception as error:
199             print error
200             sys.exit()
201
202         query = protocol_codes.request_tokenV
203         self.tls_sock_a.sendall(query)
204
205         response = self.tls_sock_a.recv(size)
206         TokenV = response.replace(protocol_codes.challenge_TokenV,"")
207         print "Challenge TokenV = " + TokenV
208         self.dump_in_file(self.tokenPS_TokenV_file,self.tokenPS + "&" +
                TokenV)
209         self.group_sign_document(self.tokenPS_TokenV_sig_file, self.
                tokenPS_TokenV_file)
210         signature = self.get_file_content(self.tokenPS_TokenV_sig_file)
211         print "Signature answer of challenge = " + signature
212         self.tls_sock_a.sendall(protocol_codes.challenge_TokenV_response +
```

```
                self.tokenPS + "&" + signature)
213         response = self.tls_sock_a.recv(size)
214
215         if protocol_codes.activated_tokenV not in response:
216             print "couldn't receive tokenV"
217             return False
218         else:
219             self.tokenV = TokenV
220             self.dump_in_file(self.tokenV_file,self.tokenV)
221             return True
222
223
224     def cast_vote(self,vote,tokenV):
225         try:
226             self.tls_sock_v.connect((self.host_v,self.port_v))
227         except Exception as error:
228             print "An error has occured"
229             print error
230             sys.exit()
231
232         composed_vote = vote + "&" + tokenV
233
234         self.dump_in_file(self.composed_vote_file,composed_vote)
235         self.group_sign_document(self.signed_composed_vote_file,self.
                composed_vote_file)
236         signed_composed_vote = self.get_file_content(self.
                signed_composed_vote_file)
237
238         print "Signed vote = " + signed_composed_vote
239
240         query = protocol_codes.vote+composed_vote + "&" +
                signed_composed_vote
241         self.tls_sock_v.sendall(query)
242         response = self.tls_sock_v.recv(size)
243         if response == protocol_codes.invalid_vote_id:
244             return -2
245         elif response == protocol_codes.tokenV_already_voted:
246             return -1
247         else:
248             return 0
249
250 if __name__ == "__main__":
251
252     if len(sys.argv) < 9 or len(sys.argv) > 10:
253         print "Usage: " + sys.argv[0] + " <host_g1> <port_g1> <host_A> <
                port_A> <host_V> <port_V> <member_key_file> <group_key_file> [<
                vote_id>]"
254         sys.exit(0)
255     host_g1 = sys.argv[1]
256     port_g1 = int(sys.argv[2])
257     host_a = sys.argv[3]
258     port_a = int(sys.argv[4])
259     host_v = sys.argv[5]
260     port_v = int(sys.argv[6])
261     member_key_file = sys.argv[7]
262     group_key_file = sys.argv[8]
263     vote = None
264     if len(sys.argv) == 10:
265         vote = sys.argv[9]
266     client = VotingClient(host_g1, port_g1, host_a, port_a, host_v, port_v,
            member_key_file, group_key_file)
267
268     print "Welcome to the voting system, before proceeding please double
```

```
            check that the configuration parameters are correct"
269     print "In this system you will be able to cast your vote for the
            elections, and make an accounting to verify that the process was made
             correctly"
270     print "Keep in mind that in order to do the accounting you must have
            casted your vote before hand"
271     valid = False
272     option = None
273     while not valid:
274         print "What would you like to do?"
275         print "1)Cast your vote"
276         print "2)Make vote Accounting"
277         print "3)Exit voting system"
278         option = input()
279         try:
280             option = int(option)
281         except Exception as error:
282             option = -1
283         if option == 1 or option == 2 or option == 3:
284             valid = True
285     if option == 1:
286         if vote is None:
287             print "vote_id needed for this option"
288             sys.exit()
289         ret = None
290         if os.path.isfile(client.tokenPS_file):
291             print "TokenPS File detected, recovering TokenPS from it"
292             client.get_tokenPS_from_file()
293         else:
294             try:
295                 ret = client.get_new_tokenPS()
296             except Exception as error:
297                 print error
298                 sys.exit()
299             if ret != protocol_codes.activated_tokenPS:
300                 sys.exit()
301
302         print "TokenPS acquired = " + client.tokenPS
303         print "Attempting to get tokenV"
304
305         if os.path.isfile(client.tokenV_file):
306             print "TokenV File detected, recovering TokenV from it to use it
                    to vote"
307             client.get_tokenV_from_file()
308         else:
309             try:
310                 client.get_new_tokenV()
311             except Exception as error:
312                 print error
313                 sys.exit()
314         print "TokenV acquired = " + client.tokenV
315         print "Attempting to Vote"
316
317         if os.path.isfile(client.vote_proof_file):
318             print "Vote Proof file detected, which means you have already
                    voted"
319         else:
320             ret = client.cast_vote(vote,client.tokenV)
321             if ret == -2:
322                 print "Invalid Vote identifier"
323             elif ret == -1:
324                 print "You already voted"
325             elif ret == 0:
```

```
326                 print "Vote casted and saved succesfully"
327     elif option == 2:
328         print "Performing an accounting"
329 
330         if os.path.isfile(client.tokenPS_file):
331             print "TokenPS file detected, running accounting of tokenPS"
332 
333             if client.tokenPS is None:
334                 client.get_tokenPS_from_file()
335             if client.tokenPS_sig is None:
336                 if os.path.isfile(client.tokenPS_sig_file):
337                     client.get_tokenPS_sig_from_file()
338                 else:
339                     client.group_sign_document(client.tokenPS_sig_file,
                            client.tokenPS_file)
340                     client.get_tokenPS_sig_from_file()
341             if client.tokenPS_sig_proof is None:
342                 if os.path.isfile(client.tokenPS_sig_proof_file):
343                     client.get_tokenPS_sig_proof_from_file()
344                 else:
345                     client.claim_signature(client.tokenPS_sig_proof_file,
                            client.tokenPS_sig_file)
346                     client.get_tokenPS_sig_proof_from_file()
347             #auditoria tokenps con g1
348             print "TokenPS for the accounting = " + client.tokenPS
349             print "Signature proof for the accounting = " + client.
                    tokenPS_sig_proof
350             if client.tokenPS_accounting(client.tokenPS,client.
                    tokenPS_sig_proof):
351                 print "Auditoria de TokenPS satisfactoria"
352             else:
353                 print "Auditoria de TokenPS no satisfactoria"
354             #fin de auditoria con g1
355             if os.path.isfile(client.tokenPS_TokenV_sig_file) and os.path.
                    isfile(client.tokenV_file):
356                 #auditoria tokenV con A
357                 client.get_tokenV_from_file()
358                 client.claim_signature(client.tokenPS_TokenV_sig_proof_file,
                        client.tokenPS_TokenV_sig_file)
359                 proof = client.get_file_content(client.
                        tokenPS_TokenV_sig_proof_file)
360                 print "Vote Proof for the accounting = " + proof
361 
362                 if client.tokenV_accounting(client.tokenV,proof):
363                     print "Auditoria de TokenV satisfactoria"
364                 else:
365                     print "Auditoria de TokenV no satisfactoria"
366                 if os.path.isfile(client.signed_composed_vote_file):
367                     proof_signed_composed_vote = None
368 
369                     if not os.path.isfile(client.
                            proof_signed_composed_vote_file):
370                         client.claim_signature(client.
                                proof_signed_composed_vote_file,client.
                                signed_composed_vote_file)
371 
372                     proof_signed_composed_vote = client.get_file_content(
                            client.proof_signed_composed_vote_file)
373 
374                     print "Proof of vote = " + proof_signed_composed_vote
375                     if client.vote_accounting(client.tokenV,
                            proof_signed_composed_vote):
376                         print "Auditoria de voto satisfactoria"
```

```
377                     else:
378                         print "Auditoria de voto no satisfactoria"
379                 else:
380                     print "No se ha encontrado prueba de voto"
381             else:
382                 print "No se ha encontrado prueba de TokenV"
383         else:
384             print "No se ha encontrado el fichero de TokenPS"
385     elif option == 3:
386         sys.exit()
```

O.2. Código Server G1

```
1  import protocol_codes
2
3  from Crypto import Random
4  from Crypto.PublicKey import RSA
5  import mysql.connector
6  from mysql.connector import errorcode
7  import random
8  import re
9  import socket
10 import ssl
11 import string
12 import subprocess
13 import sys
14 import threading
15
16
17 dbconfig = {
18   'user': 'root',
19   'password': 'password',
20   'host': '127.0.0.1',
21   'database': 'G1_DB',
22   'raise_on_warnings': True,
23 }
24
25 insert_statement = ("INSERT INTO Tokens (TokenPS, GroupSign, Id) "
26                     "VALUES (%s, %s, %s)")
27
28 select_tokenPS_statement = ("SELECT TokenPS FROM Tokens "
29                             "WHERE TokenPS = %s")
30 select_tokenPS_groupSign_from_id = ("SELECT TokenPS,GroupSign FROM Tokens "
31                             "WHERE Id = %s")
32 select_groupSign_from_tokenPS = ("SELECT GroupSign FROM Tokens "
33                             "WHERE TokenPS = %s")
34
35 class ThreadedTLSServer(object):
36     def __init__(self, host, port, keyfile, certfile, group_key_file,
       manager_key_file, crl, gml):
37         self.keyfile = keyfile
38         self.certfile = certfile
39         self.port = port
40         self.group_key_file = group_key_file
41         self.manager_key_file = manager_key_file
42         self.crl = crl
43         self.gml = gml
44         self.tokenPS_file = "tokenPS_file.txt"
45         self.mutex = threading.Semaphore()
46         self.cnx = None
47         if not self.connect_to_database():
48             print "Couldn't connect to Database"
```

```
49              raise Exception(("Error","Database connection error"))
50          self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
51          self.host = host
52          print "Host="+self.host
53          self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
54          self.sock.bind((self.host, self.port))
55
56      def connect_to_database(self):
57          try:
58              self.cnx = mysql.connector.connect(**dbconfig)
59          except mysql.connector.Error as err:
60              if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
61                  print("Something is wrong with your user name or password")
62              elif err.errno == errorcode.ER_BAD_DB_ERROR:
63                  print("Database does not exist")
64              else:
65                  print(err)
66              return False
67          else:
68            return True
69      def listen(self):
70          self.sock.listen(5)
71          self.tls_sock = ssl.wrap_socket(self.sock, ssl_version=ssl.
                PROTOCOL_TLSv1, cert_reqs=ssl.CERT_NONE, server_side=True,
                keyfile=self.keyfile, certfile=self.certfile)
72          while True:
73              print "Awaiting Connection"
74              client, address = self.tls_sock.accept()
75              print "Connection Started"
76              client.settimeout(10)
77              threading.Thread(target = self.listenToClient,args = (client,
                    address)).start()
78
79      def get_new_tokenPS(self):
80          print "Generating TokenPS"
81          tokenPS = ''.join(random.SystemRandom().choice(string.
                ascii_uppercase + string.digits) for _ in range(128))
82          while self.is_tokenPS_validated(tokenPS):
83              tokenPS = ''.join(random.SystemRandom().choice(string.
                    ascii_uppercase + string.digits) for _ in range(128))
84          return tokenPS
85
86      def is_tokenPS_validated(self,tokenPS):
87          print "Validating tokenPS"
88          cursor = self.cnx.cursor()
89          cursor.execute(select_tokenPS_statement,(tokenPS,))
90          cursor.fetchall()
91          count = cursor.rowcount
92          cursor.close()
93          return count > 0
94
95      def is_signature_revoked(self,tokenPS_sign):
96          print "checking revoked signature"
97          tmp_sign = open("tmp_sign.txt","w")
98          tmp_sign.write(tokenPS_sign)
99          tmp_sign.close()
100
101         p = subprocess.Popen(['./tools/trace','CPY06','tmp_sign.txt',self.
                group_key_file,self.crl,self.manager_key_file,self.gml],stdout=
                subprocess.PIPE)
102         p.wait()
103
104         result = p.stdout.readline()
```

```
105
106         print "RESULT == " + result
107
108         return not "VALID" in result
109
110     def store_tokenPS(self,tokenPS,tokenPS_sign,identity):
111         if self.is_tokenPS_validated(tokenPS):
112             return False
113         else:
114             cursor = self.cnx.cursor()
115             try:
116                 cursor.execute(insert_statement,(tokenPS,tokenPS_sign,
                        identity))
117             except Exception as ex:
118                 print ex
119             self.cnx.commit()
120             cursor.close()
121             return True
122
123     def listenToClient(self, client, address):
124         size = 4096
125         print "Listening to Client"
126         while True:
127             try:
128                 option = client.recv(size)
129                 print "Option: "+option
130
131                 if option == protocol_codes.request_tokenPS:
132                     self.mutex.acquire()
133                     tokenPS = self.get_new_tokenPS()
134                     print "Generated TokenPS = " + tokenPS
135                     client.sendall(protocol_codes.tokenPS + tokenPS)
136                     tokenPS_sign = client.recv(size)
137                     print "Signature received = " + tokenPS_sign
138
139                     tmp_token = open("tmp_tokenps.txt","w")
140                     tmp_token.write(tokenPS)
141                     tmp_token.close()
142
143                     if self.is_signature_revoked(tokenPS_sign) == True:
144                         self.mutex.release()
145                         client.sendall(protocol_codes.revoked_user)
146                         client.shutdown(socket.SHUT_RDWR)
147                         client.close()
148                         raise Exception('Error','Revoked User')
149
150                     print "Verifying Signature Correctness"
151                     p = subprocess.Popen(['./tools/verify','CPY06','tmp_sign
                            .txt','tmp_tokenps.txt',self.group_key_file],stdout=
                            subprocess.PIPE)
152                     p.wait()
153                     result = p.stdout.readline()
154                     print "Verification result:" + result
155                     if "VALID" not in result:
156                         self.mutex.release()
157                         client.send(protocol_codes.invalid_group_signature)
158                         client.shutdown(socket.SHUT_RDWR)
159                         client.close()
160                         raise Exception('Error','Wrong Signature')
161
162                     print "Revoking Signature Issuer"
163
164                     p = subprocess.Popen(['./tools/revoke','CPY06','tmp_sign
```

```
                    .txt',self.group_key_file,self.manager_key_file,self.
                    gml,self.crl],stdout=subprocess.PIPE)
165                 p.wait()
166
167
168                 p = subprocess.Popen(['tail',self.crl,'--lines','1'],
                        stdout=subprocess.PIPE)
169                 p.wait()
170                 identity = p.stdout.readline()
171                 matchObj = re.match('[0-9]+',identity)
172                 identity = matchObj.group()
173                 print "Issuer Identity = " + identity
174                 identity = int(identity)
175
176                 self.store_tokenPS(tokenPS,tokenPS_sign, identity)
177                 self.mutex.release()
178                 client.sendall(protocol_codes.activated_tokenPS)
179                 client.shutdown(socket.SHUT_RDWR)
180                 client.close()
181                 print "TokenPS given to voter"
182
183             elif protocol_codes.validate_tokenPS in option:
184                 tokenPS = option.replace(protocol_codes.validate_tokenPS
                        ,"")
185                 #saneando la entrada eliminando los caracteres no
                        alfanumericos, precaucion sql injection
186                 tokenPS = re.sub(r'\W+','',tokenPS)
187                 print "tokenPS to validate:"+tokenPS
188                 self.mutex.acquire()
189                 if self.is_tokenPS_validated(tokenPS):
190                     print "VALID"
191                     client.sendall(protocol_codes.valid_tokenPS)
192                 else:
193                     client.sendall(protocol_codes.invalid_tokenPS)
194                 print "RELEASING MUTEX"
195                 self.mutex.release()
196                 client.shutdown(socket.SHUT_RDWR)
197                 print "Closing connection"
198                 client.close()
199             elif protocol_codes.validate_tokenPS_and_signature in option
                    :
200                 subs = option.replace(protocol_codes.
                        validate_tokenPS_and_signature,"")
201                 splitted = subs.split("&")
202                 tokenPS = splitted[0]
203                 print "Received TokenPS = " + tokenPS
204                 token_sig = splitted[1]
205                 print "Received Signature = " + token_sig
206                 self.mutex.acquire()
207                 if not self.is_tokenPS_validated(tokenPS):
208                     print "TOKENPS IS NOT VALID"
209                     self.mutex.release()
210                     client.sendall(protocol_codes.invalid_tokenPS)
211                     client.shutdown(socket.SHUT_RDWR)
212                     client.close()
213                 elif not self.is_signature_revoked(token_sig):
214                     print "SIGNATURE IS NOT REVOKED"
215                     self.mutex.release()
216                     client.sendall(protocol_codes.non_revoked_user)
217                     client.shutdown(socket.SHUT_RDWR)
218                     client.close()
219                 else:
220                     print "Verified"
```

```
221                         self.mutex.release()
222                         client.sendall(protocol_codes.
                                valid_tokenPS_and_signature)
223                         client.shutdown(socket.SHUT_RDWR)
224                         client.close()
225                         return True
226                 elif protocol_codes.tokenPS_accounting in option:
227                     subs = option.replace(protocol_codes.tokenPS_accounting,
                            "")
228                     splitted = subs.split("&")
229                     tokenPS = splitted[0]
230                     proof = splitted[1]
231                     print "Received TokenPS = " + tokenPS
232                     print "Received Proof = " + proof
233                     cursor = self.cnx.cursor()
234                     self.mutex.acquire()
235                     cursor.execute(select_groupSign_from_tokenPS,(tokenPS,))
236                     rows = cursor.fetchall()
237                     if cursor.rowcount == 0:
238                         self.mutex.release()
239                         client.sendall(protocol_codes.wrong_voter_id)
240                         client.shutdown(socket.SHUT_RDWR)
241                         client.close()
242                         return False
243                     groupSign = rows[0][0]
244                     print "Groupsign = " + groupSign
245                     proof_file = "tmp_proof.txt"
246                     tmp_proof = open(proof_file,"w")
247                     tmp_proof.write(proof)
248                     tmp_proof.close()
249                     tmp_sign_file = "tmp_sign.txt"
250                     tmp_sign = open(tmp_sign_file,"w")
251                     tmp_sign.write(groupSign)
252                     tmp_sign.close()
253                     p = subprocess.Popen(['./tools/claim_verify','CPY06',
                            proof_file,self.group_key_file,tmp_sign_file],stdout=
                            subprocess.PIPE)
254                     p.wait()
255                     self.mutex.release()
256                     result = p.stdout.readline()
257                     print "Resultado auditoria TokenPS: " + result
258                     if "VALID" in result:
259                         client.sendall(protocol_codes.accounting_ok)
260                     else:
261                         client.sendall(protocol_codes.accounting_wrong)
262                     client.shutdown(socket.SHUT_RDWR)
263                     client.close()
264                     return True
265
266
267                 elif protocol_codes.recover_tokenPS in option:
268                     subs = option.replace(protocol_codes.recover_tokenPS,"")
269                     identity = -1
270                     try:
271                         identity = int(subs)
272                     except Exception as error:
273                         client.sendall(protocol_codes.wrong_voter_id)
274                         client.shutdown(socket.SHUT_RDWR)
275                         client.close()
276                         return False
277                     cursor = self.cnx.cursor()
278                     self.mutex.acquire()
279                     cursor.execute(select_tokenPS_groupSign_from_id,(
```

```
                        identity,))
280                     rows = cursor.fetchone()
281                     if row is None:
282                         self.mutex.release()
283                         client.sendall(protocol_codes.wrong_voter_id)
284                         client.shutdown(socket.SHUT_RDWR)
285                         client.close()
286                         return False
287                     tokenPS = row[0]
288                     groupSign = row[1]
289                     client.sendall(groupSign)
290                     proof = client.recv(4096)
291
292
293                     proof_file = "tmp_proof.txt"
294                     tmp_proof = open(proof_file,"w")
295                     tmp_proof.write(proof)
296                     tmp_proof.close()
297
298                     tmp_sign_file = "tmp_sign.txt"
299                     tmp_sign = open(tmp_sign_file,"w")
300                     tmp_sign.write(groupSign)
301                     tmp_sign.close()
302
303                     p = subprocess.Popen(['./tools/claim_verify','CPY06',
                            proof_file,self.group_key_file,tmp_sign_file],stdout=
                            subprocess.PIPE)
304                     p.wait()
305                     self.mutex.release()
306                     result = p.stdout.readline()
307
308                     print "RESULT == " + result
309                     if "VALID" in result:
310                         client.sendall(protocol_codes.tokenPS + tokenPS)
311                     else:
312                         client.sendall(protocol_codes.wrong_proof)
313                     client.shutdown(socket.SHUT_RDWR)
314                     client.close()
315                     return True
316                 else:
317                     client.sendall(protocol_codes.unknown_option + option)
318                     client.shutdown(socket.SHUT_RDWR)
319                     client.close()
320
321             except:
322                 return False
323
324
325 if __name__ == "__main__":
326
327     if len(sys.argv) != 9:
328         print "Usage: " + sys.argv[0] + " <host> <port> <private_key_file> <
                certificate> <group_key_file> <manager_key_file> <crl> <gml>"
329         sys.exit(0)
330     host = sys.argv[1]
331     port = int(sys.argv[2])
332     private_key_file = sys.argv[3]
333     certificate_file = sys.argv[4]
334     group_key_file = sys.argv[5]
335     manager_key_file = sys.argv[6]
336     crl = sys.argv[7]
337     gml = sys.argv[8]
338     my_server = ThreadedTLSServer(host, port, private_key_file,
```

```
            certificate_file, group_key_file, manager_key_file, crl, gml)
339     print "Server created succesfully"
340     my_server.listen()
```

O.3. Código Server A

```
1  import protocol_codes
2
3  from Crypto import Random
4  from Crypto.PublicKey import RSA
5  import mysql.connector
6  from mysql.connector import errorcode
7  import random
8  import re
9  import socket
10 import ssl
11 import string
12 import subprocess
13 import sys
14 import threading
15
16 dbconfig = {
17   'user': 'root',
18   'password': 'password',
19   'host': '127.0.0.1',
20   'database': 'A_DB',
21   'raise_on_warnings': True,
22 }
23
24
25 select_tokenV_statement = ("SELECT TokenV FROM Tokens "
26                            "WHERE TokenV = %s")
27 select_tokenPS_statement = ("SELECT TokenPS FROM Tokens "
28                             "WHERE TokenPS = %s")
29
30 insert_statement = ("INSERT INTO Tokens (TokenV,TokenPS, GroupSign)"
31                 "VALUES (%s,%s,%s)")
32
33 select_tokenV_groupSign_from_tokenPS = ("SELECT TokenV,GroupSign FROM Tokens
       "
34                             "WHERE TokenPS = %d")
35
36 select_groupSign_from_tokenV = ("SELECT GroupSign FROM Tokens "
37                             "WHERE TokenV = %s")
38
39 class ThreadedTLSServer(object):
40     def __init__(self, host_local, port_local, host_g1, port_g1, keyfile,
          certfile,group_key_file):
41         self.keyfile = keyfile
42         self.certfile = certfile
43         self.activated_tokenV_list = {}
44         self.mutex = threading.Semaphore()
45         self.host_local = host_local
46         self.port_local = port_local
47         self.host_g1 = host_g1
48         self.port_g1 = port_g1
49         self.group_key_file = group_key_file
50         self.cnx = None
51         self.tmp_file_message = "tmp_file_message.txt"
52         self.tmp_file_message_signature = "tmp_file_message_signature.txt"
53         if not self.connect_to_database():
54             print "Error conecting to database"
```

```
55          self.sock_voter = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
56          self.sock_voter.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,
                1)
57          self.sock_voter.bind((self.host_local, self.port_local))
58          self.sock_g1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
59          self.tls_sock_g1 = ssl.wrap_socket(self.sock_g1, ssl_version=ssl.
                PROTOCOL_TLSv1, cert_reqs=ssl.CERT_NONE)
60
61      def connect_to_database(self):
62          try:
63              self.cnx = mysql.connector.connect(**dbconfig)
64          except mysql.connector.Error as err:
65              if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
66                  print("Something is wrong with your user name or password")
67              elif err.errno == errorcode.ER_BAD_DB_ERROR:
68                  print("Database does not exist")
69              else:
70                  print(err)
71              return False
72          else:
73            return True
74
75      def listen(self):
76          self.sock_voter.listen(5)
77          self.tls_sock_voter = ssl.wrap_socket(self.sock_voter, ssl_version=
                ssl.PROTOCOL_TLSv1, cert_reqs=ssl.CERT_NONE, server_side=True,
                keyfile=self.keyfile, certfile=self.certfile)
78          while True:
79              print "Awaiting Connection"
80              client, address = self.tls_sock_voter.accept()
81              print "Connection Started"
82              client.settimeout(10)
83              threading.Thread(target = self.listenToClient,args = (client,
                    address)).start()
84
85
86      def dump_in_file(self, file_name, content):
87          tmp_file = open(file_name,"w")
88          tmp_file.write(content)
89          tmp_file.close()
90          return
91
92      def get_new_tokenV(self):
93          tokenV = ''.join(random.SystemRandom().choice(string.ascii_uppercase
                + string.digits) for _ in range(128))
94          print tokenV
95          while self.is_tokenV_validated(tokenV):
96              tokenV = ''.join(random.SystemRandom().choice(string.
                    ascii_uppercase + string.digits) for _ in range(128))
97          return tokenV
98
99      def has_tokenPS_tokenV(self,tokenPS):
100         if self.activated_tokenV_list.has_key(tokenPS):
101             return True
102         cursor = self.cnx.cursor()
103         cursor.execute(select_tokenPS_statement,(select_tokenPS_statement,))
104         cursor.fetchall()
105         count = cursor.rowcount
106         cursor.close()
107         return cursor.rowcount > 0
108
109     def is_tokenV_validated(self,tokenV):
110         if self.activated_tokenV_list.has_key(tokenV):
```

```
111             print "tokenv is validated"
112             return True
113         else:
114             cursor = self.cnx.cursor()
115             cursor.execute(select_tokenV_statement,(tokenV,))
116             cursor.fetchall()
117             count = cursor.rowcount
118             cursor.close()
119 
120             return cursor.rowcount > 0
121 
122     def store_tokenV(self,tokenV,tokenPS,tokenPS_sign):
123         if self.is_tokenV_validated(tokenV):
124             return False
125         else:
126             print "Storing TokenV entry"
127             self.activated_tokenV_list[tokenPS]=tokenV
128             cursor = self.cnx.cursor()
129             cursor.execute(insert_statement,(tokenV,tokenPS,tokenPS_sign))
130             self.cnx.commit()
131             cursor.close()
132             print "TokenV entry stored succesfully"
133             return True
134 
135     def validate_voter_g1(self, tokenPS, tokenPS_sign):
136         print "validating voter"
137         if self.sock_g1 is None:
138             self.sock_g1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
139             self.tls_sock_g1 = ssl.wrap_socket(self.sock_g1, ssl_version=ssl
                    .PROTOCOL_TLSv1, cert_reqs=ssl.CERT_NONE)
140 
141         try:
142             self.tls_sock_g1.connect((self.host_g1,self.port_g1))
143         except Exception as error:
144             print "An error has occured"
145             print error
146             sys.exit()
147         print "Requesting Voter Validation = " + protocol_codes.
                validate_tokenPS_and_signature+tokenPS+"&"+tokenPS_sign
148         self.tls_sock_g1.sendall(protocol_codes.
                validate_tokenPS_and_signature+tokenPS+"&"+tokenPS_sign)
149         response = self.tls_sock_g1.recv(4096)
150         print "response = " + response
151         self.tls_sock_g1.shutdown(socket.SHUT_RDWR)
152         self.tls_sock_g1.close()
153         self.sock_g1.close()
154         self.sock_g1 = None
155         self.tls_sock_g1 = None
156         return response
157 
158     def get_tokenV_and_signature_from_tokenPS(tokenPS):
159         cursor = self.cnx.cursor()
160         cursor.execute(select_tokenV_groupSign_from_tokenPS,(tokenPS,))
161         rows = cursor.fetchone()
162         if row is None:
163             return (None,None)
164         tokenV = row[0]
165         groupSign = row[1]
166         return (TokenV,groupSign)
167 
168     def get_tokenV_list_as_string(self):
169         return ','.join(map(str,self.activated_tokenV_list))
170 
```

```
171     def listenToClient(self, client, address):
172         size = 4096
173         print "Listening to Client"
174         while True:
175             try:
176                 option = client.recv(size)
177                 print "Option = " + option
178                 if protocol_codes.request_tokenV in option:
179                     print "Client is Requesting TokenV"
180                     subs = option.replace(protocol_codes.request_tokenV,"")
181 
182                     TokenV = self.get_new_tokenV()
183                     client.sendall(protocol_codes.challenge_TokenV + TokenV)
184                     print "Challenge of TokenV = " + TokenV
185                     query = client.recv(size)
186                     print "Response of Challenge = " + query
187                     if not protocol_codes.challenge_TokenV_response in query
                            :
188                         client.sendall(protocol_codes.wrong_proof)
189                         client.shutdown(socket.SHUT_RDWR)
190                         client.close()
191                         return False
192 
193                     msg = query.replace(protocol_codes.
                            challenge_TokenV_response,"")
194                     splitted = msg.split("&")
195                     TokenPS = splitted[0]
196                     Signature = splitted[1]
197                     print "TokenPS = " + TokenPS
198                     print "Signature = " + Signature
199                     self.mutex.acquire()
200                     print "Validating Signature"
201                     try:
202                         self.dump_in_file(self.tmp_file_message,TokenPS + "&
                                " + TokenV)
203                         self.dump_in_file(self.tmp_file_message_signature,
                                Signature)
204                         p = subprocess.Popen(['./tools/verify','CPY06',self.
                                tmp_file_message_signature,self.tmp_file_message,
                                self.group_key_file],stdout=subprocess.PIPE)
205                         p.wait()
206                         result = p.stdout.readline()
207                     except Exception as err:
208                         print err
209                     print "Result = " + result
210                     if not "VALID" in result:
211                         self.mutex.release()
212                         client.sendall(protocol_codes.wrong_proof)
213                         client.shutdown(socket.SHUT_RDWR)
214                         client.close()
215                         return False
216                     else:
217                         ret = self.validate_voter_g1(TokenPS,Signature)
218                         if ret != protocol_codes.valid_tokenPS_and_signature
                                :
219                             self.mutex.release()
220                             client.sendall(ret)
221                             client.shutdown(socket.SHUT_RDWR)
222                             client.close()
223                             return False
224                         else:
225                             self.store_tokenV(TokenV,TokenPS,Signature)
226                             self.mutex.release()
```

```
227                         client.sendall(protocol_codes.activated_tokenV)
228                         client.shutdown(SHUT_RDWR)
229                         client.close()
230                         return True
231
232                 elif protocol_codes.recover_tokenV in option:
233                     subs = option.replace(protocol_codes.recover_tokenV,"")
234                     splitted = subs.split("&")
235                     tokenPS = splitted[0]
236                     proof = splitted[1]
237                     tokenV, groupSign = self.
                            get_tokenV_and_signature_from_tokenPS(tokenPS)
238                     if (tokenV is None) and (groupsig is None):
239                         client.sendall(protocol_codes.unknown_tokenPS)
240                         client.shutdown(socket.SHUT_RDWR)
241                         client.close()
242                         return True
243
244                     self.mutex.acquire()
245                     proof_file = "tmp_proof.txt"
246                     self.dump_in_file(proof_file,proof)
247
248                     tmp_sign_file = "tmp_sign.txt"
249                     self.dump_in_file(tmp_sign_file,groupSign)
250
251                     p = subprocess.Popen(['./tools/claim_verify','CPY06',
                            proof_file,self.group_key_file,tmp_sign_file],stdout=
                            subprocess.PIPE)
252                     p.wait()
253                     self.mutex.release()
254                     result = p.stdout.readline()
255
256                     if "VALID" in result:
257                         client.sendall(protocol_codes.tokenV + tokenV)
258                     else:
259                         client.sendall(protocol_codes.wrong_proof)
260
261                     client.shutdown(socket.SHUT_RDWR)
262                     client.close()
263                     return True
264                 elif protocol_codes.tokenV_accounting in option:
265
266                     subs = option.replace(protocol_codes.tokenV_accounting,"
                            ")
267                     splitted = subs.split("&")
268                     tokenV = splitted[0]
269                     proof = splitted[1]
270                     cursor = self.cnx.cursor()
271                     self.mutex.acquire()
272
273                     cursor.execute(select_groupSign_from_tokenV,(tokenV,))
274                     rows = cursor.fetchall()
275                     if cursor.rowcount == 0:
276                         self.mutex.release()
277                         client.sendall(protocol_codes.wrong_voter_id)
278                         client.shutdown(socket.SHUT_RDWR)
279                         client.close()
280                         return False
281                     groupSign = rows[0][0]
282                     print "Groupsign = " + groupSign
283
284                     tmp_proof_file = "tmp_proof.txt"
285                     self.dump_in_file(tmp_proof_file,proof)
```

```
286
287                     tmp_sign_file = "tmp_sign.txt"
288                     self.dump_in_file(tmp_sign_file,groupSign)
289
290                     try:
291                         p = subprocess.Popen(['./tools/claim_verify','CPY06'
                                ,tmp_proof_file,self.group_key_file,tmp_sign_file
                                ],stdout=subprocess.PIPE)
292                         p.wait()
293                     except Exception as err:
294                         print err
295                     self.mutex.release()
296                     result = p.stdout.readline()
297                     print result
298                     if "VALID" in result:
299                         client.sendall(protocol_codes.accounting_ok)
300                     else:
301                         client.sendall(protocol_codes.accounting_wrong)
302                     client.shutdown(socket.SHUT_RDWR)
303                     client.close()
304                     return True
305
306                 elif option == "get TokenV List":
307                     self.mutex.acquire()
308                     list = self.get_tokenV_list_as_string()
309
310             except:
311                 return False
312
313
314 if __name__ == "__main__":
315
316     if len(sys.argv) != 8:
317         print "Usage: " + sys.argv[0] + " <host_local> <port_local> <host_g1
                > <port_g1> <private_key_file> <certificate> <group_key_file>"
318         sys.exit(0)
319     host_local = sys.argv[1]
320     port_local = int(sys.argv[2])
321     host_g1 = sys.argv[3]
322     port_g1 = int(sys.argv[4])
323     private_key_file = sys.argv[5]
324     certificate_file = sys.argv[6]
325     group_key_file = sys.argv[7]
326     my_server = ThreadedTLSServer(host_local, port_local, host_g1, port_g1,
            private_key_file, certificate_file,group_key_file)
327     print "Server created succesfully"
328     my_server.listen()
```

O.4. Código Server V

```
1  import protocol_codes
2
3  from Crypto import Random
4  from Crypto.PublicKey import RSA
5  import base64
6  import hashlib
7  import json
8  import mysql.connector
9  from mysql.connector import errorcode
10 import random
11 import re
12 import socket
```

```
13 import ssl
14 import string
15 import subprocess
16 import sys
17 import threading
18
19 dbconfig = {
20   'user': 'root',
21   'password': 'password',
22   'host': '127.0.0.1',
23   'database': 'V_DB',
24   'raise_on_warnings': True,
25 }
26
27
28 select_tokenV_accounting_statement = ("SELECT GroupSign from Votes "
29                 "WHERE TokenV = %s")
30 insert_statement = ("INSERT INTO Votes (Vote,TokenV,GroupSign,Valid) "
31                 "VALUES (%s,%s,%s,%s)")
32 validate_vote_statement = ("UPDATE Votes SET Valid=1 "
33                 "WHERE TokenV = %s")
34 select_tokenV_statement = ("SELECT TokenV from Votes "
35                 "WHERE TokenV = %s")
36 remove_invalid_votes_statement = ("DELETE FROM VOTES "
37                                   "WHERE Valid=0")
38 vote_count_statement = ("SELECT Vote, COUNT(*) "
39                 "From Votes WHERE Valid=1 GROUP BY Vote")
40
41
42 class ThreadedTLSServer(object):
43     def __init__(self, host_local, port_local, host_a, port_a, keyfile,
        certfile, group_key_file):
44         self.keyfile = keyfile
45         f=open(self.keyfile,'r')
46         self.private_key = RSA.importKey(f.read())
47         f.close()
48         self.certfile = certfile
49         self.mutex = threading.Semaphore()
50         self.host_local = host_local
51         self.port_local = port_local
52         self.host_a = host_a
53         self.port_a = port_a
54         self.cnx = None
55         self.group_key_file = group_key_file
56         self.tmp_file_message_signature = ".tmp_file_message_signature.txt"
57         self.tmp_file_message = ".tmp_file_message.txt"
58         self.tmp_group_sign_file = ".tmp_group_sign_file.txt"
59         self.tmp_group_sign_proof_file = ".tmp_group_sign_proof_file.txt"
60         if not self.connect_to_database():
61             print "Error conecting to database"
62         self.sock_voter = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
63         self.sock_voter.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,
            1)
64         self.sock_voter.bind((self.host_local, self.port_local))
65         self.sock_a = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
66         self.tls_sock_a = ssl.wrap_socket(self.sock_a, ssl_version=ssl.
            PROTOCOL_TLSv1, cert_reqs=ssl.CERT_NONE)
67         self.tokenV_List = None
68         self.vote_data = None
69
70         self.load_voting_data("candidates.json")
71
72     def connect_to_database(self):
```

```
73          try:
74              self.cnx = mysql.connector.connect(**dbconfig)
75          except mysql.connector.Error as err:
76              if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
77                  print("Something is wrong with your user name or password")
78              elif err.errno == errorcode.ER_BAD_DB_ERROR:
79                  print("Database does not exist")
80              else:
81                  print(err)
82              return False
83          else:
84            return True
85
86
87      def listen(self):
88          self.sock_voter.listen(5)
89          self.tls_sock_voter = ssl.wrap_socket(self.sock_voter, ssl_version=
                ssl.PROTOCOL_TLSv1, cert_reqs=ssl.CERT_NONE, server_side=True,
                keyfile=self.keyfile, certfile=self.certfile)
90          while True:
91              print "Awaiting Connection"
92              client, address = self.tls_sock_voter.accept()
93              print "Connection Started"
94              client.settimeout(10)
95              threading.Thread(target = self.listenToClient,args = (client,
                    address)).start()
96
97
98      def load_voting_data(self,candidates_file):
99          f = open(candidates_file,'r')
100         raw_data = f.read()
101         self.vote_data = json.loads(raw_data)
102         f.close()
103         return
104
105     def dump_in_file(self, file_name, content):
106         tmp_file = open(file_name,"w")
107         tmp_file.write(content)
108         tmp_file.close()
109         return
110
111     def is_tokenV_validated(self,tokenV):
112         if self.tokenV_List is None:
113             return True
114         else:
115             return tokenV in self.tokenV_List
116
117
118     def generate_vote_dig_sig(self,vote,tokenV,):
119         string = vote + tokenV
120         digest = hashlib.sha512(string).hexdigest()
121         print digest
122         print type(digest)
123         signature = base64.standard_b64encode(self.private_key.encrypt(
                digest,32)[0])
124         print signature
125         return signature
126
127     def count_votes(self):
128         cursor = self.cnx.cursor()
129         cursor.execute(vote_count_statement)
130         rows = cursor.fetchall()
131         result = {}
```

```
132         for row in rows:
133             result[row[0]] = row[1]
134         return result
135 
136     def store_vote(self,vote,tokenV,signature):
137         print "Storing vote"
138         cursor = self.cnx.cursor()
139         cursor.execute(insert_statement,(vote,tokenV,signature,0))
140         self.cnx.commit()
141         cursor.close()
142         return True
143 
144     def vote_accounting(self,tokenV,proof):
145         cursor = self.cnx.cursor()
146         cursor.execute(select_tokenV_accounting_statement,(tokenV,))
147         rows = cursor.fetchall()
148         if not rows:
149             return False
150         groupSign = rows[0][0]
151         self.mutex.acquire()
152         self.dump_in_file(self.tmp_group_sign_file,groupSign)
153         self.dump_in_file(self.tmp_group_sign_proof_file,proof)
154         try:
155             p = subprocess.Popen(['./tools/claim_verify','CPY06',self.
                    tmp_group_sign_proof_file,self.group_key_file,self.
                    tmp_group_sign_file],stdout=subprocess.PIPE)
156             p.wait()
157         except Exception as err:
158             print err
159         self.mutex.release()
160         result = p.stdout.readline()
161         if "VALID" in result:
162             return True
163         else:
164             return False
165 
166 
167 
168     def has_tokenV_voted(self,tokenV):
169         print "Checking if next TokenV has voted: " + tokenV
170         cursor = self.cnx.cursor()
171         cursor.execute(select_tokenV_statement,(tokenV,))
172         cursor.fetchall()
173         if cursor.rowcount > 0:
174             cursor.close()
175             print "TokenV Has Voted"
176             return True
177         else:
178             cursor.close()
179             print "TokenV hasnt voted"
180             return False
181 
182     def validate_stored_votes(self):
183         for tokenV in self.tokenV_List:
184             cursor = self.cnx.cursor()
185             if cursor is None:
186                 self.cnx.commit()
187                 return False
188             cursor.execute(validate_vote_statement,(tokenV,))
189             cursor.close()
190         self.cnx.commit()
191         return True
192 
```

```
193     def remove_invalid_votes(self):
194         cursor = self.cnx.cursor()
195         cursor.execute(remove_invalid_votes_statement)
196         cursor.close()
197         self.cnx.commit()
198         return True
199 
200     def is_voted_id_valid(self,vote_id):
201         for x in self.vote_data['candidates']:
202             if x['Id'] == vote_id:
203                 print "Vote for "+x['Name']
204                 return True
205         return False
206 
207     def check_valid_signature(self,message,signature):
208         tmp_file = open(self.tmp_file_message,"w")
209         tmp_file.write(message)
210         tmp_file.close()
211         tmp_file = open(self.tmp_file_message_signature,"w")
212         tmp_file.write(signature)
213         tmp_file.close()
214         print "Verifying Signature"
215         p = subprocess.Popen(['./tools/verify','CPY06',self.
                tmp_file_message_signature,self.tmp_file_message,self.
                group_key_file],stdout=subprocess.PIPE)
216         p.wait()
217         result = p.stdout.readline()
218         print "Verification result = " + result
219         return "VALID" in result
220 
221     def listenToClient(self, client, address):
222         size = 4096
223         print "Listening to Client"
224         while True:
225             try:
226                 option = client.recv(size)
227                 print "Option = " + option
228                 if protocol_codes.vote in option:
229                     print "Vote Request Received"
230                     tmp = option.replace(protocol_codes.vote,"")
231                     splitted = tmp.split("&")
232                     vote_id = splitted[0]
233                     tokenV = splitted[1]
234                     signature = splitted[2]
235                     print "TokenV = " + tokenV
236                     print "Vote_id = " + vote_id
237                     print "Signature = " + signature
238                     self.mutex.acquire()
239                     print "Mutex acquired"
240                     if self.check_valid_signature(vote_id + "&" + tokenV,
                            signature):
241                         if not self.is_voted_id_valid(int(vote_id)):
242                             self.mutex.release()
243                             print "Invalid vote id"
244                             client.sendall(protocol_codes.invalid_vote_id)
245                             return False
246 
247                         if self.has_tokenV_voted(tokenV):
248                             self.mutex.release()
249                             client.sendall(protocol_codes.
                                    tokenV_already_voted)
250                         else:
251                             self.mutex.release()
```

```
252                                 #dig_sign = self.generate_vote_dig_sig(vote_id,
                                        tokenV)
253                                 self.store_vote(vote_id,tokenV,signature)
254                                 client.sendall(protocol_codes.vote_ok)
255                         else:
256                             self.mutex.release()
257                             client.sendall(protocol_codes.
                                    invalid_group_signature)
258
259                         client.shutdown(socket.SHUT_RDWR)
260                         client.close()
261                     elif protocol_codes.vote_accounting in option:
262                         print "Vote Accounting Received"
263                         tmp = option.replace(protocol_codes.vote_accounting,"")
264                         splitted = tmp.split("&")
265                         tokenV = splitted[0]
266                         proof = splitted[1]
267                         print "TokenV = " + tokenV
268                         print "Proof = " + proof
269                         if self.vote_accounting(tokenV,proof):
270                             print "Accounting went Ok"
271                             client.sendall(protocol_codes.accounting_ok)
272                         else:
273                             print "Accounting went Wrong"
274                             client.sendall(protocol_codes.wrong_proof)
275                         client.shutdown(socket.SHUT_RDWR)
276                         client.close()
277
278                 except:
279                     return False
280
281
282 if __name__ == "__main__":
283
284     if len(sys.argv) != 8:
285         print "Usage: " + sys.argv[0] + " <host_local> <port_local> <host_a>
                <port_a> <private_key_file> <certificate> <group_key_file>"
286         sys.exit(0)
287     host_local = sys.argv[1]
288     port_local = int(sys.argv[2])
289     host_a = sys.argv[3]
290     port_a = int(sys.argv[4])
291     private_key_file = sys.argv[5]
292     certificate_file = sys.argv[6]
293     group_key_file = sys.argv[7]
294     my_server = ThreadedTLSServer(host_local, port_local, host_a, port_a,
            private_key_file, certificate_file, group_key_file)
295     print "Server created succesfully"
296     my_server.listen()
```

O.5. Código Protocol Codes

```
1  request_tokenPS = "#1"
2  activated_tokenPS = "#2"
3  valid_tokenPS = "#3"
4  validate_tokenPS = "#4:"
5  validate_tokenPS_and_signature = "#5:"
6  valid_tokenPS_and_signature = "#6"
7  unknown_option = "#7:"
8  tokenPS = "#8:"
9  request_tokenV = "#9:"
10 preactivated_tokenV = "#10:"
```

```
activated_tokenV = "#11:"
recover_tokenPS = "#12:"
recover_tokenV = "#13:"
tokenV = "#14:"
tokenPS_accounting = "#15:"
tokenV_accounting = "#16:"
vote = "#17:"
vote_identifier = "#18:"
vote_accounting = "#19:"
accounting_ok = "#20"
tokenV_already_voted = "#21"
vote_ok = "#22"
challenge_TokenV = "#23:"
challenge_TokenV_response = "#24:"

revoked_user = "&1"
invalid_group_signature = "&2"
invalid_tokenPS = "&3"
non_revoked_user = "&4"
invalid_voter = "&5"
wrong_voter_id = "&6"
wrong_proof = "&7"
unknown_tokenPS = "&8"
invalid_vote_id = "&9"
```